1988-12

A conceptual design of a Software Base Management System for the Computer Aided Prototyping System

Galik, Daniel

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

A CONCEPTUAL DESIGN OF A
SOFTWARE BASE MANAGEMENT SYSTEM
FOR THE
COMPUTER AIDED PROTOTYPING SYSTEM

by

Daniel Galik

December 1988

Thesis Advisor: Luqi

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE

1a Report Security Classification: Unclassified	1b Restrictive Markings
2a Security Classification Authority	3 Distribution Availability of Report: Approved for public release; distribution is unlimited
2b Declassification/Downgrading Schedule	
4 Performing Organization Report Number(s)	5 Monitoring Organization Report Number(s)
6a Name of Performing Organization: Naval Postgraduate School; 6b Office Symbol (*If Applicable*): 62	7a Name of Monitoring Organization: Naval Postgraduate School
6c Address (*city, state, and ZIP code*): Monterey, CA 93943-5000	7b Address (*city, state, and ZIP code*): Monterey, CA 93943-5000
8a Name of Funding/Sponsoring Organization; 8b Office Symbol (*If Applicable*)	9 Procurement Instrument Identification Number
8c Address (*city, state, and ZIP code*)	10 Source of Funding Numbers: Program Element Number; Project No; Task No; Work Unit Accession No

11 Title (*Include Security Classification*): A Conceptual Design of a Software Base Management System for the Computer Aided Prototyping System

12 Personal Author(s): Galik, Daniel

13a Type of Report	13b Time Covered	14 Date of Report (*year, month,day*)	15 Page Count
Master's Thesis	From To	1988 December	81

16 Supplementary Notation: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 Cosati Codes			18 Subject Terms (*continue on reverse if necessary and identify by block number*)
Field	Group	Subgroup	Ada, Rapid Prototyping, Software Reusability, Object-Oriented Database Management Systems, Computer Aided Prototyping System, Prototype System Description Language, CAPS, PSDL

19 Abstract (*continue on reverse if necessary and identify by block number*

This thesis builds upon work previously done in the development of the Computer Aided Prototyping System (CAPS) and the Prototype System Description Language (PSDL), and presents a conceptual design for the Software Base Management System (SBMS) component of CAPS. The SBMS is the most critical component of CAPS as it will coordinate the retrieval and integration of Ada software modules. A robust SBMS that enables a software system designer to successfully retrieve reusable Ada components will expedite the prototype development process and enhance designer productivity. Implementation of the conceptual design will be the basis for further work in this area. (Ada is a registered trademark of the United States Government, Ada Joint Program Office.)

20 Distribution/Availability of Abstract	21 Abstract Security Classification	
[X] unclassified/unlimited [ ] same as report [ ] DTIC users	Unclassified	
22a Name of Responsible Individual: Luqi	22b Telephone (*Include Area code*): (408) 646-2735	22c Office Symbol: 52LQ

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

security classification of this page: Unclassified

Approved for public release; distribution is unlimited

A CONCEPTUAL DESIGN OF A SOFTWARE BASE MANAGEMENT SYSTEM FOR THE COMPUTER AIDED PROTOTYPING SYSTEM

by

Daniel Galik
Lieutenant Commander, United States Navy
B.S., United States Naval Academy, 1976

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1988

ABSTRACT

This thesis builds upon work previously done in the development of the Computer Aided Prototyping System (CAPS) and the Prototype System Description Language (PSDL), and presents a conceptual design for the Software Base Management System (SBMS) component of CAPS. The SBMS is the most critical component of CAPS as it will coordinate the retrieval and integration of Ada software modules. A robust SBMS that enables a software system designer to successfully retrieve reusable Ada components will expedite the prototype development process and enhance designer productivity. Implementation of the conceptual design will be the basis for further work in this area. (Ada is a registered trademark of the United States Government, Ada Joint Program Office.)

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. BACKGROUND

Creating software for hard real-time and embedded computer systems is an immense and complex job, and involves increasingly high costs. The rapid advances in hardware technology of the past ten years have not been accompanied by similar advances in software development technologies. The primary problem manifesting itself was the reliability of software. Programming is a costly, labor-intensive activity, and there has been a steady increase in the proportion of system development costs allocated to software development. Errors made early in the development cycle of large, complex systems contributed significantly to poor reliability and resulted in large software maintenance costs.

The United States Department of Defense (DOD) is the world's largest user of computers. The mounting problem in software development is of critical concern to DOD for two reasons. First, is the fact that software reliability directly affects overall weapon system reliability, where the cost of system failure could be immense, not only in terms of equipment failure but also in terms of human lives. The second reason involves DOD budget constraints, particularly in today's environment where every defense dollar spent is subject to intense public scrutiny and justification. Software costs are becoming the principal factor in embedded computer costs. It is also known that embedded systems software costs are by far the major contributor to the overall software budget within the DOD [Ref. 1:p. 30]. A report issued in 1976 that contributed to the development of Ada summarized some of the problems of software development, stating that the problems "...appear in the form of software that is nonresponsive to user needs, unreliable,

excessively expensive, untimely, inflexible, difficult to maintain, and not reusable." [Ref 1:p. 41]

1. Hard Real-Time and Embedded Software Systems

The programming of large, hard real-time and embedded computer software systems is a very difficult and complex task for system designers. In a typical Navy weapons system, there may be hundreds of computer programs running simultaneously controlling and coordinating inputs from components such as shipboard or aircraft navigation, radar, and weapons fire control systems. In 1987, after 12 years of design, review, and development, the DOD mandated the use of Ada as its single, common computer programming language for computers integral to weapons systems. The mandate, through Directive 3405.2, required that Ada be used in all embedded computer software programs, whether new programs or upgrades to existing ones [Ref. 2:p. 33]. This particular domain of embedded systems was targeted by DOD because it offers a potential for enormous savings in system life cycle costs.

A hard real-time system can be defined as a software or firmware controlled system that performs all its process functions within critical specified time constraints [Ref. 3:p. 3]. The majority of embedded systems are real-time systems, particularly typical Navy weapons systems, where there is a requirement for real-time processing, and the system must respond to physical stimuli in real time. A real-time system usually includes a set of independent hardware devices that operate at widely differing speeds that must be controlled and synchronized. A real-time system is much more difficult to design and implement than a non-real-time system. As discussed in Ref. 3, some of the difficulties include:

- Handling of stringent time requirements and performance specifications.
- Interfacing with a real-time clock.

- Control of hardware devices such as communication lines, terminals, and computer resources.
- Processing of messages that arrive at irregular intervals, with fluctuating input rates, and with different priorities.
- Control of fault conditions with facilities for various degrees of recovery.
- Handling of queues and buffers for storage of messages and data items.
- Modeling of concurrent conditions into a proper set of concurrent processes.
- Allocation and control of concurrent processes to processors.
- Handling of communication and synchronization between concurrent processes.
- Protection of data shared between concurrent processes.
- Scheduling and dispatching (including priority handling) of concurrent processes.

Typical operation of hard real-time systems is that they are expected to run continuously in an automated fashion with extreme reliability. They are also required to perform rapid processing to meet critical real-time constraints in satisfying system performance requirements. In tactical weapon systems, system reliability and the ability to meet critical time constraints are of utmost importance. The development of these complex, hard real-time software systems remains a costly operation and requires more work at the front end of the development cycle in properly identifying and analyzing all the critical time constraint relationships.

2. Integrated Programming Environments

With the demand for hard real-time and embedded computer systems increasing, it is becoming critical that new approaches be proposed for the development of large software systems. Typically, these large embedded systems have similar requirements for critical real-time control and high reliability. Software engineers and end-users would benefit from an automated methodology which allows validation of design specifications and functional requirements early in the development life cycle. A fast, efficient, easy-to-use tool would increase designer productivity and also allow the user to feel more confident that the final product is feasible. Current programming languages alone cannot deal

adequately with the growing complexity of large computer programming projects, particularly those involving embedded systems. Fundamental changes are needed in the way we program. Just as high-level languages enabled the programmer to escape from the intricacies of machine-level code, higher-level programming systems can provide help in understanding and manipulating complex systems and components [Ref. 4:p. 391]. As these new programming tools continue to develop and mature, they will provide another level of abstraction in the programming process, making life easier for the designer of large, complex systems.

Well-designed integrated programming environments which incorporate advanced programming tools, highly interactive editors, and high resolution graphics, will have a positive impact on the future of programming and software development. Integrated programming environments can also be classified as "computer-aided design systems for software" [Ref. 5:p.xi]. Their goal is to provide a unified programming environment with a large set of tools that stimulates program conception at a high level of abstraction. Research is rapidly expanding in the field of programming environments with the intent of improving long term cost-effective software reliability by providing a computer-based capability to support the entire process of program design, development, test, and maintenance.

3. Rapid Prototyping and the Computer Aided Prototyping System

The Computer Aided Prototyping System (CAPS) is a currently conceptualized tool which is being developed with a goal of improving software development technology [Ref. 6]. CAPS will aid the software designer in the requirements analysis of hard, real-time systems by using specifications and reusable software components to automate the rapid prototyping process. It implements the rapid prototyping concept utilizing a high level prototyping language called Prototype System Description Language (PSDL)

[Ref. 7]. The component elements that make up CAPS will be discussed in Chapter II. One of the most critical components of CAPS is the software database and its associated Software Based Management System (SBMS). A robust SBMS that enables a software system designer to successfully retrieve reusable software components will expedite the prototype development process and enhance designer productivity.

The goal of rapid prototyping is to develop an executable model of the intended system. When utilized during the early stages of the development life cycle, rapid prototyping allows validation of the requirements, specifications, and initial design, before valuable time and effort are expended on implementation software. Rapid prototyping initially establishes an iterative process between the user and the designer to concurrently define specifications and requirements for the time critical aspects of the envisioned system. The designer then constructs a model or prototype of the system in a high level, prototyping language (PSDL). This prototype is a partial representation of the system, including only those critical attributes necessary for meeting user requirements, and is used as an aid in analysis and design rather than as production software [Ref. 7:pp. 2-5]. During demonstrations of the prototype, the user validates the prototype's actual performance against its expected performance. If the prototype fails to execute properly or to meet any critical timing constraints, the user identifies required modifications and redefines the critical specifications and requirements (see Fig. 1). This process continues until the user and the designer both agree that the prototype successfully meets the time critical aspects of the envisioned system. A key aspect of the process is the feedback from the user to the designer. This iterative communication process should result in a model that will ultimately meet the intended requirements of the user. Following this final validation, the designer uses the prototype as a basis for the design and eventual hand coding of the production software. The design of large scale Ada software systems typically have

particularly strict requirements on accuracy, safety, and reliability. These are difficult to meet without extensive prototyping. A rapid prototyping environment for creating and modifying an executable prototype is needed. The PSDL language, its associated prototyping method, and programming environment apply well to the modeling and design of hard real-time embedded Ada software systems.

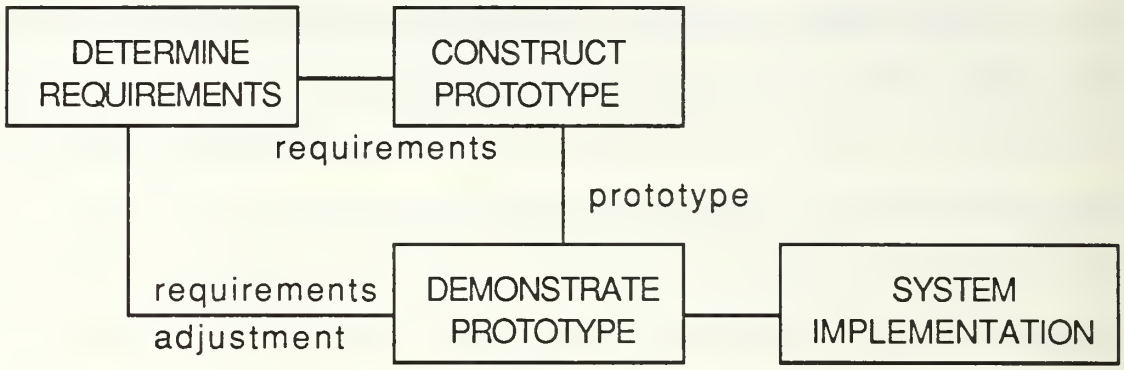

Figure 1. Process of Requirements Determination and Validation

B. OBJECTIVES

The objective of this thesis is the development of a conceptual level design for the Software Base Management System of the Computer Aided Prototyping System. This conceptual design will be the basis for further research leading to full implementation of the Software Base Management System.

C. ORGANIZATION

Chapter II will contain a detailed discussion of the role of all the CAPS system components. A survey of recent work in the area of software reusability with emphasis on Ada will also be presented. This chapter will also include an overview and survey of research involved with database management techniques with primary focus being on those that make use of software libraries, and perform the classification, selection, and retrieval of software components. The actual conceptual level design for the CAPS SBMS will be

presented in Chapter III. Conclusions and recommendations will be presented in Chapter IV.

II. PREVIOUS RESEARCH AND SURVEY OF SOFTWARE REUSABILITY

A. PREVIOUS RESEARCH

The Software Base Management System is one element of the Computer Aided Prototype System (CAPS) tool. An overview of CAPS is presented here in greater detail to provide an overall framework for the Software Base Management System.

1. The Computer Aided Prototyping System

The Computer Aided Prototyping System (CAPS) relies on a number of major software tools (see Fig. 2) to assist the designer in constructing and executing a prototype. First, the computer-aided environment includes a Software Base Management System which creates uniform retrieval specifications for Ada software modules in the software database and later retrieves these reusable modules for assembling the executable prototype. Second, a graphics-capable user interface including a syntax-directed editor expedites the designer's data entry at a terminal and prevents syntax errors in the design. Finally, an execution support system demonstrates and measures the prototype's performance and analyzes the accuracy of design specifications. Rapid construction of the prototype relies on application of the rapid prototyping methodology along with a support environment which automates the steps involved. [Ref. 8:pp. 2-10]

2. Prototype System Description Language

PSDL was designed as the primary connection between the designer and the components of the CAPS. By definition, PSDL is a high-level prototyping language with special features appropriate for defining critical real-time constraints, and is applied at the specification or design stage [Ref. 9:pp. 3, 23]. PSDL was specifically designed for the

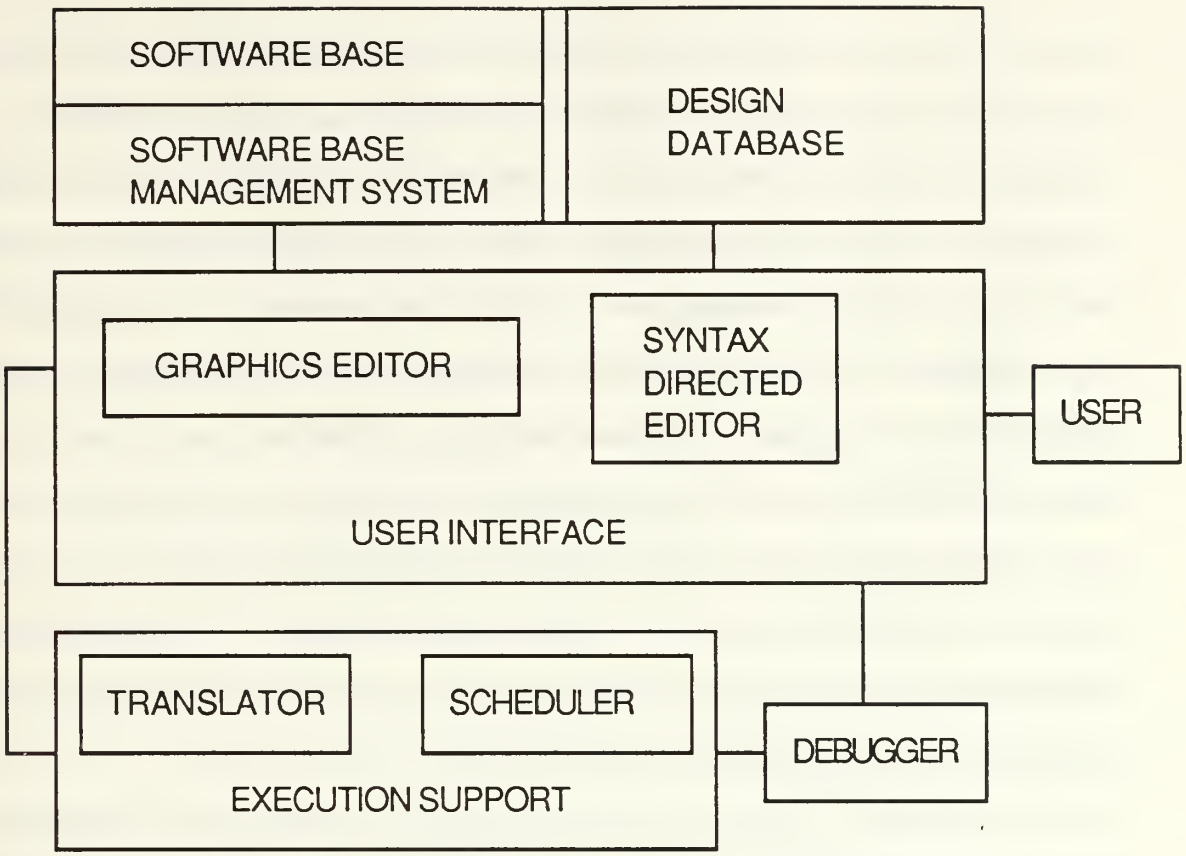

Figure 2. Computer Aided Prototyping System

modeling of embedded and hard real-time software. PSDL and CAPS use Ada as an implementation language. The Department of Defense has mandated the requirement that the language Ada be used for the design of hard real-time embedded systems. Ada supports key software engineering principles including software portability and reuse. However, it is not perfect, nor is it the ultimate in programming languages [Ref. 10: p. xiv]. Ada constructs do not support real-time modeling directly, while PSDL constructs do provide effective techniques and tools to specify timing constraints. PSDL has selected and transformed the good language features of Ada primitive constructs into a small and simple set of PSDL language constructs, which is convenient for the designer of the prototype model. It is simpler to describe the structure of a system and the

relation between system components in PSDL than in Ada, since PSDL allows a designer to express his thoughts and ideas clearly, easily, and significantly faster at a specification or a design level with notations based on abstractions. The important points are that the software tools and the prototyping method of PSDL lead to a well structured prototype and that the resulting PSDL prototype is executable. PSDL components can be mapped into Ada directly. The methodology involves the use of specification and prototyping languages during the early design phase, and then use a language such as Ada as the implementation language for the final product . Ada is a large and powerful programming language. It is a good underlying programming or implementation language for PSDL. However, it is too hard and too cumbersome to use as a design language directly. The mapping between PSDL and Ada and the use of reusable Ada components are the keys to making PSDL prototypes executable and useful in large Ada projects. [Ref. 7:pp. 51-53]

In order to rapidly construct a prototype, PSDL aids the designer in systematically refining and decomposing each critical component into its lower level components. Uniform PSDL specifications associated with each lower level description act as templates for retrieving reusable Ada software components having similar specifications from the CAPS software database. Thus, the use of PSDL produces a computational model consisting of the basic building blocks needed to describe the abstractions and concepts of the hierarchically structured prototype. The PSDL execution support environment then verifies the design and the validity of the prototype's real-time requirements. The actual execution of the prototype model demonstrates whether these critical timing constraints will perform in an acceptable manner that meets the timing constraints of the system as a whole [Ref. 7:pp. 2-7]. Complete details concerning PSDL are provided in Ref. 7.

3. The PSDL Prototyping Method

The rapid construction of a prototype in PSDL is made possible by the associated prototyping method and support environment. The support environment reduces the efforts of the designer by automating some of the tasks involved in prototype construction. A PSDL prototype is constructed as a hierarchy of subsystems, referred to as the components of the prototype. Each component is either a reusable module available in the software base or is defined in terms of the PSDL computational model. The code of the prototype usually cannot be used in the final implementation because the prototype is not a complete representation of the intended system. PSDL was designed to support an automated environment for rapid prototyping. The most important parts of the environment are a static scheduler, a translator, a dynamic scheduler, a software based management system, and a syntax directed editor. The editor can provide a user-friendly interface and significantly reduce the amount of effort required of the designer. A graphical interface with a high resolution display and a pointer device is convenient for manipulating the enhanced data flow diagrams. The purpose of the PSDL prototyping method and its support environment is the rapid construction of executable prototypes for large real-time systems. The PSDL prototyping method develops a hierarchically structured design by a process of stepwise refinement, guided by the computational model and the software base.

In the iterative rapid prototyping environment, the initial version of the prototype will meet some but not all of the requirements, and will provide an initial structure from which to work. A prototype is evaluated and tested through execution by the designer. Once the designer believes that the prototype meets all the requirements, the prototype is demonstrated for the user. Identified faults are reviewed along with the specifications, and the iterative process of refinement continues. The advantage of PSDL and its associated automated prototyping tool is that this iterative refinement process can proceed quickly and

ultimately should result in a model of a more reliable system, that the user agrees will meet the requirements. Hard real-time constraints of embedded systems can be more effectively modeled using PSDL.

4. Benefits of CAPS

CAPS and the use of PSDL offer the potential of serving as a valuable software development tool specifically for hard real-time systems. The prototyping process is speeded up by automation and by reducing the conceptual burden of the analyst and prototype designer. PSDL was designed to interface to an automated support environment with a software base containing reusable software components. PSDL has constructs for recording timing constraints, for defining operator and data abstractions, and for specifying non-procedural control constraints. The most important aspects of real-time systems design are enforcing maximum response time and data synchronization constraints. PSDL handles both of these aspects well. The execution support system allows the designer to check the feasibility of a set of real-time constraints by monitoring and evaluating the execution of the prototype model. CAPS offers a software tool that is a most practical way to support rapid prototyping of hard real-time software systems. This together with the features of PSDL for large scale software design make it a good candidate for inclusion in an advanced Ada programming environment. An experienced PSDL user should be able to construct a prototype significantly faster than an experienced Ada user [Ref. 7:p. 52]. The user can validate the hard real-time system constraints by executing the constructed prototype. Further research efforts are currently in progress to implement all the subsystem components of the automated support environment of PSDL.

B. SURVEY OF SOFTWARE REUSABILITY

1. Introduction

The concept of software reusability has been the subject of a great deal of research, especially during the past five years. The repetitive nature of computer programming would lead one to conclude that many of the same software modules have been written again and again by different programmers through the years. The potential benefits and advantages of reusability include the following:

- Increased programmer productivity and efficiency
- Lower overall development costs
- Improved software maintenance and quality
- Faster software development

These potential benefits become larger when reusability is applied to the development of complex, real-time, and embedded systems, because the intellectual and development effort that goes into designing and implementing these complex systems is much larger than for more simple systems. This section of the thesis will review some of the more significant work which has been accomplished in the area of research of software reusability. Applicability to the CAPS software base of reusable components will also be analyzed.

2. Classification and Retrieval

One of the critical issues in reusing software components is the problem of locating and retrieving them from a large collection. Prieto-Diaz and Freeman note that the proper classification of reusable software components is central to making code reusability an attractive approach to software development [Ref. 11:p. 6]. A good discussion of the Prieto-Diaz and Freeman classification scheme is presented in Ref. 11, and some of the highlights of that article will now be reviewed. They propose an integrated classification scheme that is embedded in a retrieval system and supported by an evaluation mechanism.

The purpose of the evaluation mechanism is to help users discriminate among very similar components in the software base, and allow the user to select the components that require the least amount of conversion effort. Prieto-Diaz and Freeman present the following algorithm to illustrate the code reuse process proposed in their model [Ref. 11:p. 7]:

```
begin
   search library
   if identical match then terminate
   else
      collect similar components
      for each component
         compute degree of match
      end
      rank and select best
      modify component
   endif
end.
```

The components in the software base are described by standard attributes that capture their functional characteristics. It should also be noted that exact matches are rare and the retrieved code will most likely have to be adapted and modified to suit the user's needs. Prieto-Diaz notes that existing classification schemes are usually very general and are not organized by reusability-related attributes. He proposes a faceted scheme where a software component is described by a tuple of six terms where each term is an attribute value of a selected facet. Classifying a component consists of selecting the sextuple that best describes the component. Each component is described by a sextuple that is maintained in a relational database system. The evaluation mechanism allows the user to select the attributes which are most important, which will then be used with metrics in the evaluation process. Prieto-Diaz has conducted some preliminary testing of his model on a small scale with favorable results. It appears to offer a promising approach to reusability. Some of the classification concepts and software base management issues presented by Prieto-Diaz and Freeman have applicability to CAPS, and future results of their work are worth reviewing for potential applicability to the CAPS Software Base Management System.

3. The Graphical Object-Structured Editor

A different approach to reusability is presented in a system known as the Graphical Object-Structured Editor (GOEDIT) [Ref. 12]. This editor combines some features of a classical syntax-directed editor and integrated programming environment, in order to perform some operations on a software base consisting of software object modules. The goal of GOEDIT is to provide an automated environment for rapid prototyping system ideas including reusability. All the software modules in GOEDIT are treated as objects, with each class of objects being indicated by a text icon. A programmer will select one of the particular icons and then browse through all the instances within that class. This system has a number of deficiencies as the user must browse through all the instances of a class until he finds what he is looking for. As the software base gets larger, this will result in a lot of wasted time. Preliminary testing of the system found that it was useful for experienced programmers who were familiar with the icons and contents of the software base, but the system was not very useful for a programmer who was unfamiliar with the organization and nomenclature of the software base. The developers of GOEDIT also do not make clear exactly what criteria or methodology is used to classify the different software modules in the software base.

4. The Reusable Software Library

Noteworthy work has been accomplished at Intermetrics, Inc in the development of a Reusable Software Library (RSL) [Ref. 13]. The RSL consists of the RSL database and four subsystems:

- Library Management
- User Query
- Software Component Retrieval and Evaluation
- Software Computer-Aided Design

The foundation of the RSL is the software base which stores the attributes of each reusable software component. Originally, Intermetrics used up to as many as 60 possible attributes in describing and classifying a software component. This was later shortened to 14 attributes. The overall classification strategy was very similar to the one used in standard mathematical libraries, with the use of broad categories such as math functions, data structures, sort and search routines. The interesting component of the RSL is the Software Component Retrieval and Evaluation (SCORE) subsystem. SCORE is an interactive system that helps the user select the most appropriate software to reuse by identifying components that perform the function requested by the user. SCORE will present the user with an ordered list of recommended components. The key part of SCORE's evaluation process is the identification of the software application domain. When the user invokes SCORE, he is prompted to specify the application domain and to indicate the relative importance of each of the different software attributes. SCORE will search the software base and evaluate and rate the software components against the user specified needs. This method of evaluating components is very similar to the method proposed by Prieto-Diaz and Freeman as discussed previously. One of the deficiencies in this system occurs in the objective assignment of ratings to the attribute values of each of the software components. This is especially true in more complex application domains. The RSL places a great deal of importance on the role of the software base librarian, who is solely responsible for the proper classification and entry of components into the software base. Some type of standard technique would be needed to help librarians in maintaining consistency in rating the attributes. The emphasis on application domain may serve to limit flexibility in the software base. A similar module may belong to a number of different application domains, and it would be inefficient to store that same module in a number of different areas.

Goguen also stresses the use of application domains and notes that the organization and cataloging of a library of reusable software components is a very difficult problem [Ref. 14:p. 26]. One possible approach that he recommends is to use a hierarchical classification scheme with the use of keywords. At the highest level, the keywords might be organized by application domain. The lowest levels of the classification hierarchy might contain descriptions of program properties in forms more readable by machines than users, which Goguen feels might support a sophisticated search capability.

5. Software Factories

In this country there has been a reluctance and resistance to the actual implementation of software reusability. The Japanese have instituted the concept of a "software factory" and have achieved remarkable rates of reusability, exceeding 50 percent, with productivity gains of 20 percent per year [Ref. 15:p. 159]. The concept of software reuse is a key component of the factory organization, which actively promotes reusability. Matsumoto reports on an experimental, object-oriented retrieval system that is currently being developed [Ref. 15:p. 174]. This retrieval system is also a simple keyword based system, that will make use of a synonym library to provide a standard normalized keyword from the user input. Attempts are being made to make this system an interactive, knowledge-based type system. If software reusability is to be a success in the U.S., then greater corporate level emphasis will be required along with a restructuring of the software organization to support reusability. Freeman introduces the concept of "reusable software engineering," which has as an objective the broad reuse of all information generated in the course of development, including code reuse, design reuse, specification reuse, test plan reuse, and application model reuse [Ref. 16:p. 6]. The goal of a future software parts

technology is the development of a programming environment in which reusing code is the norm, and not the exception [Ref. 17:p. 130].

6. Object-Oriented Programming and Reusability

Johnson and Foote make an argument that object-oriented programming promotes software reuse [Ref. 18:p. 22]. One of the reasons that object-oriented programming is becoming more popular is that the concept of software reusability is becoming more important. As discussed by Johnson and Foote, object-oriented programming languages encourage software reuse in a number of ways. Class definitions provide modularity and information hiding. Late binding of procedure calls means that objects require less information about each other. In the context of object-oriented programming, polymorphism refers to the capability for different classes of objects to respond to exactly the same set of messages. A polymorphic procedure is easier to reuse than one that is not polymorphic, because it will work with a wider range of arguments. Class inheritance permits a class to be reused in a modified form by making subclasses from it. Modularity makes it easier to understand the effect of changes to a program. Polymorphism reduces the number of procedures, and thus the size of the program that has to be understood by the maintainer. Class inheritance permits a new version of a program to be built without affecting the old version.

Johnson and Foote note that the Smalltalk community practices reuse very successfully. The keys to successful software reuse are attitude, tools, and techniques. Smalltalk programmers actively seek to borrow system classes or classes invented by other programmers. They expect to spend as much time reading old code to see how to reuse it as writing new code. The Smalltalk programming environment includes a number of tools that make it easier to reuse classes, including a browser and cross reference tools.

Meyer and Interactive Software Engineering Inc. have developed an environment for reusability known as "Eiffel," that places heavy emphasis on the use of abstract data types and object-oriented design. Meyer defines object-oriented design as the construction of software systems as structured collections of abstract data-type implementations. The Eiffel language and environment consists of a design methodology, a library, and a set of supporting tools. The Eiffel library provides the basic building blocks and consists of a set of classes that implement some of the most important data structures. Inheritance plays a central role in the Eiffel environment and the language supports multiple inheritance, which is used heavily in the basic library. Eiffel implements some of the concepts of object-oriented programming with inheritance. Most of the existing object-oriented languages such as Simula and C++ support only single inheritance, while Eiffel supports multiple inheritance. [Ref. 19]

7. Reusability and Abstraction

One of the dominant themes that a number of different researchers agree with is that greater use of abstraction results in greater reusability. Abstraction and reusability are two sides of the same coin [Ref. 20:p. 37]. Parnas notes that information hiding and abstraction are key to the development of software that is well-defined and well-documented, which lends itself to reusability [Ref. 21:p. 240]. Parnas feels that by developing and cataloging software modules based on abstractions, we can greatly increase the likelihood that some of the software we write can be reused. Booch notes that the way to solve a complex software problem is to break it down into smaller, more manageable component objects [Ref. 22:p. 2]. According to Booch, by composing large systems in layers of abstraction from smaller systems that we know are correct, the user can feel confident in the correctness of the entire system.

Booch proposes a framework for classifying reusable components that can be divided up into three major groups of abstractions: structures, tools, and subsystems. A structure is a component that denotes an object or class of objects characterized as an abstract state machine or an abstract data type. A tool is a component that denotes an algorithmic abstraction targeted to an object or class of objects. A subsystem is a component that denotes a logical collection of cooperating structures and tools. Booch's taxonomy of reusable components is reviewed thoroughly in Ref. 22 and is illustrated in Figure 3.

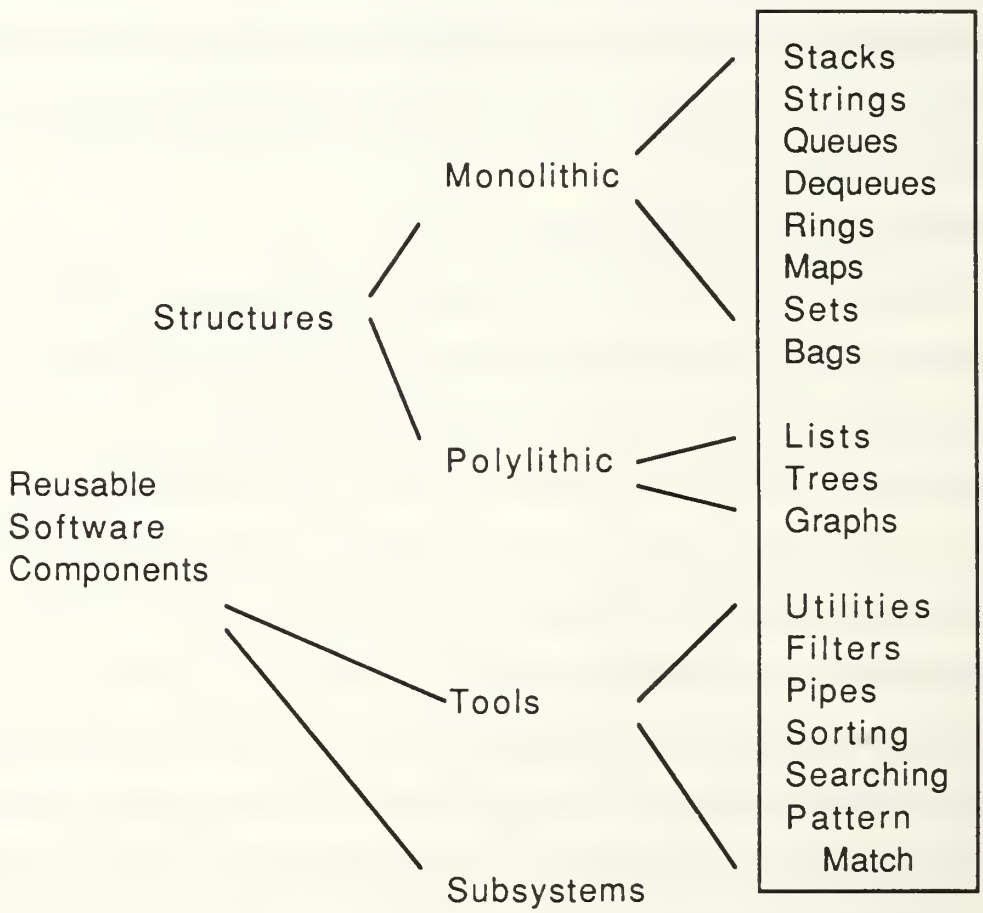

Figure 3. Booch Taxonomy

Booch's taxonomy of reusable components has become quite popular and there are a number of government agencies that are currently using the components in a library format. Booch mentions the need for a powerful data model, such as a relational database, to handle the difficult problem of component retrieval. Currently, the Booch library is maintained in a catalog type format and is not integrated with any type of database management system.

IBM has conducted work that relied heavily on abstraction in its development of a library of software "building blocks" [Ref. 23]. This particular library is oriented towards the general domain of systems programming. The IBM library contains typical abstract data type building blocks such as a stack, queue, list, and map, in addition to procedural building blocks such as message handlers and command parsers. It is interesting to note that preliminary testing has indicated that abstract data types are not as predominant in actual project use as IBM expected them to be. IBM also encountered a great deal of resistance on the part of programmers to use the reusable building block modules. The whole experiment has served to indicate to IBM the difficulties that are associated in trying to implement reuse technology in an organization that has been developing software in accordance with longstanding, proven methodologies.

8. Object-Oriented Design

Object-Oriented Design (OOD) is a new concept that will take some time for software system designers to get used to. Ada is considered an object-oriented language although it is not as object-oriented as Smalltalk and Simula since it does not support the principles of classes or inheritance. Ada was designed to be used primarily in embedded, real-time system applications. The older software system design methodologies such as Structured Analysis and Structured Design are intended primarily for non-real-time applications [Ref. 24:p. 1]. Ada is a relatively new language and was designed to include

the most recent advances in software engineering technology. These advances include such concepts as abstraction and information hiding. These desirable properties can best be fully utilized using the new concepts and methodologies of Object-Oriented Design (OOD). Ada software development efforts will result in a high quality software product if we choose to make full use of the OOD development methodology, as it best enables us to easily exploit the software engineering concepts embodied in Ada. With the exception of OOD, any existing software development methodology will have to be modified to some extent before it can be used for an Ada software development effort [Ref. 24:p. 2]. As the concepts and methodologies of OOD continue to mature, software system designers will acquire more and more expertise in its use and will most probably develop their own personal libraries consisting of Ada software object components. These libraries should provide a good base from which to construct a formalized Ada software object database that can be used within the CAPS software rapid prototyping framework.

OOD has already been successfully demonstrated on a number of different projects in the defense industry. This author recently attended a briefing by personnel from FMC Corporation, who successfully used OOD throughout the entire design and implementation efforts for the development of embedded software for the control system for the Bradley Fighting Vehicle. OOD has also been successfully used by General Electric in its Satellite Tracking System contract, and also by Ford Aerospace on an application involving 30,000 lines of code. [Ref. 25]

Here follows a brief overview of the Object-Oriented Design methodology as described by Booch [Ref. 22] and outlined by EVB Software Engineering, Inc. [Ref. 24]. OOD is a methodology which is used primarily during the design phase of the software development cycle and it deals with high level abstract objects and operations. It follows

the object orientation encouraged by such languages as Smalltalk, Simula, and Lisp. Booch states that

> ...object oriented development is an approach to software design in which the decomposition of a system is based upon the concept of an object. An object is an entity whose behavior is characterized by the actions that it suffers and that it requires of other objects. A program that implements a model of reality may thus be viewed as a set of objects that interact with one another. [Ref. 22:p. 12]

OOD is a hardware/software analysis and design method based upon data abstraction in which systems are composed or decomposed into a network of object modules. Real-time systems are viewed as a set of "black box" objects sending and receiving messages. They are also event driven systems where timing constraints and response times are of critical importance. Preliminary experience with the use of OOD on real-time systems has shown that there is a great similarity of objects used in different systems. Many of them seemed to be part of some type of control system managing timing constraints between the different objects making up the control system. This would imply that real-time Ada software development efforts using OOD have a greater potential for reusability, and also that many of these Ada software objects would be good candidates for an Ada software object database that could be incorporated within a prototyping tool such as CAPS.

As an object-oriented methodology, OOD is different in both approach and results from the traditional functional decomposition and software modeling methodologies. Object-oriented approaches do not seek to break a large problem into smaller, more solvable pieces. Instead they identify those objects and operations in the real world which must be manipulated to effect a solution to the user's problem. They deal with these items at a high level of abstraction, without regard to their low-level implementation. Meyer defines OOD as the construction of software systems as structured collections of abstract data type implementations, with every module built upon the use of

data abstractions [Ref. 19:p. 59]. OOD approaches do not construct models of the objects, but rather describe how the operations and the objects will interact in the proposed solution to the user's problems. With their emphasis on abstraction and information hiding, object modules should be easier to maintain and understand. These same characteristics should also contribute to reusability.

9. Ada and Reusability

Ada was originally designed to support the development of large programs composed of reusable software components. Each Ada compiler that is produced in industry must first be validated by the government. This validation ensures that there will not be any supersets or subsets of the language, and makes Ada an excellent language upon which to build a library of reusable components. Ada supports reusability in several ways, as discussed in Ref. 12. Ada has a rich variety of program units such as subprograms, packages, and tasks, all of which have their own syntactic interface specifications. These specifications determine how to build composite program structures and specify how those units may be reused. Ada separates the interface specification of a program unit from the implementation body. This type of information hiding provides good support for reusability. Both specification and implementation body can be reused. Ada's strong typing supports the waterfall model in the software life cycle, providing software reusability in a stepwise manner. One of the key features of Ada that supports reusability is the package concept which is a mechanism whereby one can define an abstract data type. Another key feature is the generic procedure. In some other languages you may have to write a separate routine to handle operations on elements which are different data types. With Ada generics, you don't have this problem. The design of Ada attempted to incorporate the more advanced software engineering concepts. While the language is large

and has some problems, it is still a powerful language and should provide good potential for reusability.

10. The Ada Software Repository

The Ada Software Repository (ASR) [Ref. 26] is a repository of Ada programs and software components that has been established on the Defense Data Network since 1984. The ASR is a public-domain collection which serves to provide a free source of Ada programs and information. The ASR is organized into different high level application categories such as artificial intelligence, communications, and software development tools. If a programmer desires to use one of the programs available in the ASR, he must scan through the index, and then scan each of the programs within the application area he is interested in. This type of catalog listing of reusable software is typical of most current reusable software libraries. Finding a component is a time-consuming process. The Air Force is currently working on a database management interface to the ASR, which is to be based on a relational model using SQL. Although some of the Ada components within the ASR may be useful to include within a CAPS software base, the majority of items in the ASR are large, application-specific tools, that cannot be used by CAPS in a hard, real-time, rapid prototyping environment.

11. The Common Ada Missile Packages (CAMP) Project

The CAMP project was initiated in 1984 by the Software Technology for Adaptable, Reliable Systems (STARS) Joint Program Office [Ref. 27]. The series of contracts associated with the CAMP project were performed by McDonnell Douglas Corporation. The purpose of CAMP was to demonstrate the feasibility and utility of software reuse for real-time embedded systems. A software parts library was constructed consisting of 452 operational flight software components for missile systems. The 452 CAMP software parts ranged in size from 10 to 100 Ada statements, and implemented

various missile operational flight functions. For the purposes of the CAMP project, a part is defined as a package, subprogram, or task usable in a stand-alone fashion. In other words, they may use or be used with other parts, but they do not depend on other packages, subprograms, or tasks encapsulated with them to perform some function. CAMP uses two approaches to parts identification: application and architecture. It uses an interactive expert system shell that guides missile software system designers through the software construction process. Several knowledge bases are used to contain information about Ada programming and the missile systems architecture. From September 1985 to May 1988, the CAMP parts were distributed to over 125 government agencies and contractors by the Ada Joint Program Office. The net result is that the CAMP project has been successful in demonstrating productive software reuse in building actual real-time embedded systems. The parts are now being considered for numerous government systems including NASA's space station, the advanced tactical fighter, and other new aircraft and missile systems. The CAMP software parts are available for limited distribution to government agencies. The parts should make excellent candidates for inclusion within the CAPS software base.

12. Summary

There has been a great deal of work done in the area of software reusability and much still remains to be accomplished. A number of different approaches have been tried in attempting to find the most appropriate method to classify software components, and to find a methodology for their efficient storage and retrieval from a software base. The integration of a classification, storage, and retrieval mechanism within an advanced software development tool would serve to greatly enhance software reusability. The development of a Software Base Management System for CAPS, consisting of reusable Ada components, specifically addresses these issues.

III. CONCEPTUAL DESIGN FOR THE SOFTWARE BASE MANAGEMENT SYSTEM

A. THE SELECTION OF A DATABASE MANAGEMENT SYSTEM FOR THE CAPS SOFTWARE DATABASE

1. Relational, Hierarchical, and Network Database Management Systems (DBMS)

There are a number of existing database technologies that were evaluated for inclusion into the CAPS environment. The SBMS will play a crucial role in the prototype development process as the rate of successful retrievals will directly impact the productivity rate of the designer. Current database management systems have a number of characteristics that negatively impact efficiency and productivity. These characteristics include the following:

- fixed set of structures and operations
- limitations of a fixed set of predefined types
- redundancy of data
- lack of abstraction capabilities
- schema not easily modifiable

Database management systems based on the hierarchical, network, or relational data models are characterized by a fixed set of structures and operations. Application programs must be written to interpret the data and implement the structures and operations not in the fixed set. The traditional table-oriented data model of relational database systems impose limitations and constraints as the designer is restricted to use a fixed set of predefined types. Another characteristic of relational models is a redundancy of data, as a separate physical structure is needed for each different view of the data. In the relational

model, the database schema is represented as an unstructured set of relations. This is fine if there are only a few relations in the schema but problems occur if there are many relations. The relational model has proven to work well for certain specific applications such as bank account systems and university student information systems. However in more advanced complex applications, the designer requires more flexibility. The hierarchical and network data models are powerful but they also lacked abstraction capabilities. You have to have a picture of the hierarchical structure in your mind always in order to effectively use these models, which is very difficult to do if the schema is large and complex. Once the schemas are in place, they remain fairly static and are not easily modifiable.

2. Object Technology

A relatively new database management technology is currently emerging and is focused on the concepts of object-oriented programming languages. These new object-oriented database management systems are specifically being targeted to meet the data management requirements of "complex data and process intensive applications" [Ref. 28]. Examples of complex data and process intensive applications include:

- computer aided design
- computer aided software engineering
- computer integrated manufacturing
- computer aided rapid prototyping.

All of these applications have in common large, complex, and highly interrelated data objects. Most of the operations associated with the data objects are also complex and are performed through the use of tools. Object-oriented DBMS's (OODBMS) represent the next natural step in the evolution of DBMS's. They attempt to provide facilities to define any structure and operation the user may want, not just a fixed set. This flexibility is a main advantage of OODBMS's. Since these applications are very complex, a main

objective of OODBMS's is to increase designer productivity. Abstraction is also an important capability as a very large amount of data with complex relationships cannot be manipulated by a set of complex operations unless extensive abstraction mechanisms are available.

Currently, there is not total agreement within the database community as to exactly what an object is. Each different system that comes out has slight variations as to the definition of an object. A discussion of the most common descriptions of an object follows, as discussed in Ref. 28. An object is a description of an entity in an application domain. It can be something as simple as an ordinary number, or something as complex as the assembly of parts that make up a ship or submarine. It also can be the text that makes up a software module. An object has a unique system defined identifier and a set of operations that are defined on that object. Objects which are similar are classified into classes. A class defines the property names and operations of similar objects it represents. A class itself is also viewed as an object. Class objects, as instance objects also have properties, operations, and identities. The notion of class hierarchy and inheritance of properties along the class hierarchy is another characteristic of the object-oriented data model. All subclasses of a class inherit all properties defined for the class and can have additional properties local to them. Abstraction (generalization) imposes structure on the set of classes and allows irrelevant details to be ignored. Another form of abstraction called composition, is the building of large objects using small objects. The large object is an assembly or a composite object, and the small objects are components. Examples include some of the large software object modules, which are composites that are made up of atomic component object modules.

3. Properties of OODBMS

Listed below are the more significant properties of an OODBMS:

- persistency
- active data
- extensibility
- abstraction

The fundamental properties of an OODBMS are that they must support persistency, which means that the data objects must be non-volatile while maintained within the database. OODBMS's must also support the concept of active data. The OODBMS must have an extensible set of data structures and an extensible set of operations. Extensibility means that the set of operations and structures available to database designers and users is not fixed and can be extended to meet user requirements. One other fundamental property as mentioned previously is abstraction, which includes the concepts of object composition, classification, generalization, and encapsulation. As a database management system, an OODBMS must provide the standard database functions of concurrency control, recovery, transaction management, and security. The object-oriented data model upon which the OODBMS is built provides the designer with an expressive tool for representing his objects and has the advantages of understandability, simplicity, and accessibility.

It is important to note that OODBMS's are very new and a great deal of debate and research is going on currently to determine how best to merge the concepts of object-oriented programming languages with database management technology. There currently are only a handful of commercially available OODBMS's and those that are available have not yet fully demonstrated all the fundamental concepts and theories of OODBMS's as outlined in the paragraphs above. However for the complex data applications of computer aided rapid prototyping, the use of an OODBMS for the software object database is most appropriate. It will represent a new approach to the problem of managing software databases.

B. THE SOFTWARE DATABASE

As discussed in Chapter II, the Software Base Management System (SBMS) will provide a mechanism that will enable the software system designer to successfully retrieve reusable Ada components during the rapid prototype development process. The first issue that must be dealt with in attempting to design the SBMS is the content of the database itself. In order to promote reusability and some form of structure and organization, the components that make up the database must all be inserted in accordance with a set of guidelines and prescribed criteria. The method that this author proposes is that the software database will consist of "objects," which will represent PSDL specifications, Ada software components, and associated methods that will be used to manipulate the data within the object. A more complete definition and discussion of just what an "object" is within the framework of the CAPS software base will be presented later in this Chapter. The Ada software module within the object should be a product of the Object-Oriented Design process, as described by Booch in Ref. 10. Not all Ada modules would be acceptable to be stored within the CAPS software base. Atomic level components which have been designed in accordance with the principles of OOD offer the greatest potential for reusability, and are recommended as the types of modules which would offer the greatest potential for successful use within the CAPS prototyping process. This is further explained in the next section.

1. Applying OOD to the CAPS Software Base Components

The application of OOD to the software base refers only to the design of the Ada code itself, which is just one of the attributes of an "object" in the CAPS software base. The full definition of an object in the CAPS software base includes the PSDL specifications and the operations (methods) on the object. (This full definition is given in Paragraph C.1 on page 35). The use of Booch's Object-Oriented Design methodology in developing Ada

software modules is highly recommended as being the most beneficial approach to be used in developing a software database for CAPS. This is because the OOD methodology places emphasis on developing highly cohesive and loosely coupled object modules using a high level of abstraction. To best support the rapid prototyping process, the Ada component modules must be at an atomic level with no interconnections or dependencies on other modules within the software base. Each module will be self-contained and will perform a specific function while processing a number of inputs and outputs. A major characteristic of these modules is that they will perform the required function or operation within specified timing constraints. As discussed in the survey in Chapter II, the 452 parts that make up the software library in the Common Ada Missile Package (CAMP) project would be excellent candidates for inclusion within the CAPS software base. These parts are all stand-alone modules that do not depend on other packages or subprograms to perform their specified function. While applying the OOD methodology to the CAPS software base components, there is a greater potential to achieve the following benefits:

- reducing complexity
- improving reliability
- enhancing maintainability
- encouraging object module reusability

According to Berard of EVB Inc, module coupling and module cohesion are addressed virtually automatically when using OOD. When applied correctly, OOD will almost always result in modules which are highly cohesive (performing a single specific function) and loosely coupled (weak interconnection between modules). The best examples of Ada software object modules are contained in the books by Booch (see Refs 10 and 22). Another example is the case of the typical real-time control system such as a Weapons Controller, which is illustrated in Figure 4. The components of the Weapons Controller include the control panel, missile, gun, and radar sensors. Each of these

components in the Weapons Controller is represented by an object or abstract data type, which is an Ada software module that consists of an encapsulation of data and the operations on the data. Each component object module sends messages to the other object modules utilizing the Ada interface specification for the module. The interface specification is the only way to gain access to an object's data and its operations. The details of how the Ada object module implements the operations of the Weapons Controller components is hidden within the Ada abstract data type representation for each component.

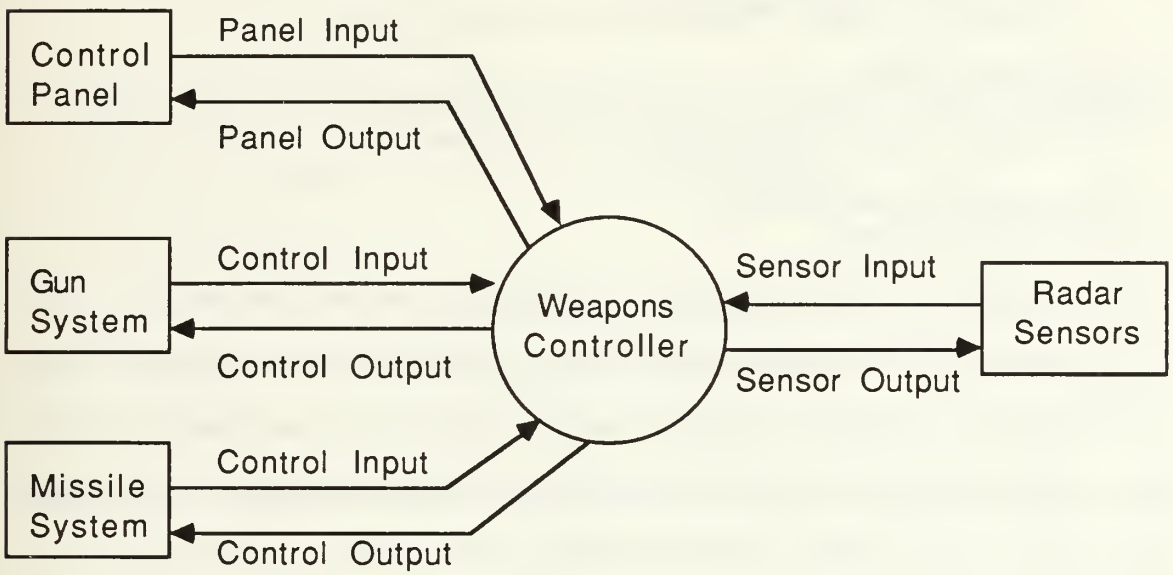

Figure 4. Weapons Controller

Within the Department of Defense, military weapons systems for the Army, Navy, or Air Force, all share similar components in that there is some type of control system that is monitoring inputs and sensors, while controlling the operation of some type of weapons delivery platform. These platforms can be based on land (such as a tank), sea (ships), or air (aircraft). Using Ada and the OOD methodology, there is a distinct potential

for reusability of some of these common objects shared by each of these systems. Some of the more common object modules included in many systems include buffers, device drivers, sorting routines, mathematical functions, and input/output of different data types. Each of these should be encapsulated within generic packages and instantiated as required. In Figure 4, the Weapons Controller can have many different component object modules associated with its different functions. One specific operation is the start-up operation which is modeled with the Ada object module listed below:

```
PROCEDURE controller_start_up (control_panel : IN real;
        sensor_input : IN real;
        missile_control : OUT real) IS

  enable_signal : real;
BEGIN
  enable_signal := calculate_control_level (control_panel,
                                sensor_input);
  missile_control := enable_signal;
END controller_start_up;
```

"Object oriented design is, in its simplest form, based on a seemingly elementary idea. Computing systems perform certain actions on certain objects; to obtain flexible and reusable systems, it is better to base the structure of software on the objects than on the actions." [Ref. 19:p. xiv] It should be stressed that software engineers should use pre-existing, proven Ada software objects when implementing code for any Ada program unit. Reusability is a key issue in the development of Ada software. Ada software objects should be designed with reusability in mind. OOD is still a fairly new concept but early results seem to indicate that it has the potential for significantly improving the quality of production software.

Methodologies such as Structured Analysis and Structured Design were created when most applications were not real-time. Also, when using the older design methodologies, software designers still had to address issues such as module coupling, module cohesion, and module size. It has also been found that large software applications

developed using Ada and OOD often require substantially less source code than applications developed in the more traditional languages. The larger the software product, the more pronounced this effect becomes [Ref. 24:p. I-8].

As the OOD methodology matures and the defense contractors acquire more experience in using it, they should be able to begin to recognize and exploit object reusability. DOD can also aid the process by specifying in its contracts that the contractor address OOD and reusability. Perhaps an award fee incentive can be provided for the contractors that can demonstrate that they have been able to reuse object modules from previous contracts. The net result should ultimately lead to eventual cost savings.

C. THE DESIGN OF AN OBJECT-ORIENTED SCHEMA FOR THE CAPS SOFTWARE BASE

1. Introduction

An "object" is an entity that represents a package of information and descriptions of how to manipulate the information or data within the object. Objects combine the properties of procedures and data, which is in contrast to conventional programming which uses separate procedures and data. Within the CAPS software base, an object consists of a number of attributes or properties which are the PSDL specifications for an Ada software module. The Ada code itself for the module is also one of the properties of the object. All of the action in an object-oriented system comes from sending messages between objects. Objects respond to messages using their methods to perform operations on the object. This message sending process supports the principle of data abstraction. The message describes what the programmer wants to happen, and does not get involved in the details of exactly how it should happen. From the outside, a programmer can only ask the object to do something by sending it a message. The method, which is internal to the object, contains all the details of how the the object's information is to be manipulated. The important thing

to note is that methods cannot be separated from the object. An example of some of the types of methods that will be associated with an object in the CAPS software base are operations to display object properties, operations to search through instances of a class looking for specific values, and operations to add, delete, or modify property values. The details of some of these operations will be discussed in the following sections of this Chapter. This thesis will attempt to demonstrate that the use of an object-oriented database management system for the CAPS software base is the best approach that can be utilized. Through the use of an object's internal methods (operations), great flexibility and power is provided in performing complex operations on the data contained within an object (PSDL specifications and Ada code).

2. PSDL Specifications and Ada Software Modules

A uniform set of PSDL specifications must be associated with each Ada software object component module that is in the software database. Each specification will be the primary means by which a reusable Ada software component will be retrieved from the database. The reusable components in the software base are used to construct subsystems of the prototype, and the available reusable components are used to guide the decomposition process by which the behavior of the prototype is refined. There are a number of attributes associated with each PSDL specification for an object module, however each attribute may not necessarily have a value associated with it for a specific module. Ada software object component retrieval based on partial matches of specified subsets of these attributes will be a key function of the SBMS. PSDL serves as the link between the software prototype system designer and the software base. When the designer gives a decomposition for a composite module, the behavior of each part of the composite module is given in the form of a PSDL specification. Further decomposition of a module is attempted only if the retrieval from the software base fails. The SBMS does the retrievals

based on approximate matching of the object module template with the specification parts of each of the objects in the software base. This is accomplished through the methods associated with the objects in the software base. For example, a method on a class of objects can search through each instance of the class looking for specific PSDL property values. The method can be written to return only exact matches or to return modules that have partial matches of just some of the PSDL specifications.

3. Software Base Classification Scheme

In object-oriented technology, a class is a description of one or more objects that are similar. Also, each object described by a class is called an instance of that class. The class describes all the similarities of its instances. Each instance contains information about the object that distinguishes it from the other instances or objects in the class. The methods that describe an object's response to messages is found within its class. All of the instances (objects) of a class use the same methods to respond to particular messages. The class itself is also an object. A class serves several purposes. It provides the description of how objects behave in response to messages, and it also provides an interface for the programmer to interact with the definitions of the objects. [Ref. 29:p. 78]

Within the CAPS software base, there are two primary classes that represent the two different kinds of PSDL components that exist. Each Ada software module component can be classified as either a PSDL "operator" or PSDL "type." PSDL operators are normally implemented in Ada as functions, state machines, and packages. The specification part of the PSDL operator contains attributes describing the form of the interface, the timing characteristics, and both formal and informal descriptions of the observable behavior of the operator. The attributes both specify the operator and form the basis for retrievals from the software base. Each instance in the class of operators or types is an object which is distinguished from every other instance in the class by the different

values associated with each of the properties (PSDL specifications) of the object. The following is a listing of possible attributes associated with a PSDL operator:

- NAME, INPUT, OUTPUT, MAXIMUM EXECUTION TIME,
- MAXIMUM RESPONSE TIME, MINIMUM CALLING PERIOD,
- EXCEPTIONS, INITIAL STATE, GENERIC, STATES,
- DESCRIPTION, PERIOD.

PSDL types correspond to Ada abstract data types and are normally implemented as Ada packages. A PSDL type can also be specified as a PSDL operator. The attributes that make up PSDL types are:

- NAME, GENERIC, OPERATOR NAME

It would be the function of the database administrator to insure that each Ada software object module is properly classified with values assigned to each of the attributes as appropriate. In the rapid prototyping environment, the database administrator role will most probably be fulfilled by the prototype designer, as he may have to write object modules for insertion into the database based on unsuccessful searches of the software object database.

4. Mapping of PSDL Specifications to Ada

Each Ada software module in the software base is uniquely identified by its associated PSDL specifications. Some of the modules may have the same name, but they will be distinguished from each other by differences in the values for their specifications. An example of a mapping of a PSDL specification to its associated Ada module is illustrated below. Additional examples of PSDL specifications and their corresponding Ada modules are included in Appendix A.

```
PSDL SPECIFICATION
OPERATOR controller_start_up
SPECIFICATION
 INPUT  control_panel : real,  sensor_input : real
 OUTPUT missile_control : real
```

```
  BY REQUIREMENTS  startup time
 MAXIMUM EXECUTION TIME  90 ms
  BY REQUIREMENTS  control_panel_max
 DESCRIPTION
 { Extracts the control panel input and sensor input and
  uses them to calculate control signal to missile.
 }
END
```

ADA MODULE CORRESPONDING TO ABOVE SPECIFICATION

```
WITH weapons_controller_package;
USE weapons_controller_package;
PROCEDURE controller_start_up (control_panel : IN real;
      sensor_input : IN real;
      missile_control : OUT real) IS
 enable_signal : real;
BEGIN
 enable_signal := calculate_control_level (control_panel,  sensor_input);
 missile_control := enable_signal;
END controller_start_up;
```

5. The Formation of the Software Base Schema

Recognizing the fact that the majority of rapid prototyping applications will involve military real-time embedded systems, the software base schema should be designed to most efficiently support the development of these hard, real-time systems. It is anticipated that the size of the software base will be quite large, therefore the software objects should be managed so that they can be stored and retrieved easily in accordance with the needs of the prototype developer.

The schema that is proposed for the CAPS software database is based on the object-oriented data model [Ref. 30:pp. 1-10]. The first step in the development of an object-oriented software base system is the specification phase and focuses on the schema formation. This phase has several goals as listed below:

- Identify and define the objects
- Identify and define any class hierarchy among the objects
- Specify the properties associated with each object
- Specify the frequent operations performed on the objects

A preliminary discussion of what an object is in the CAPS software base was discussed in Section C.1 of this Chapter. The objects in the CAPS software base are all components consisting of PSDL specifications to identify associated Ada modules. There is a class hierarchy among these components as the Ada modules can be specified by either PSDL operators or PSDL types. Although this serves to present a simple identification of the categories in the CAPS software base application domain, it should be noted that there will be many object instances associated with each category or class. This class hierarchy is illustrated below (see Figure 5):

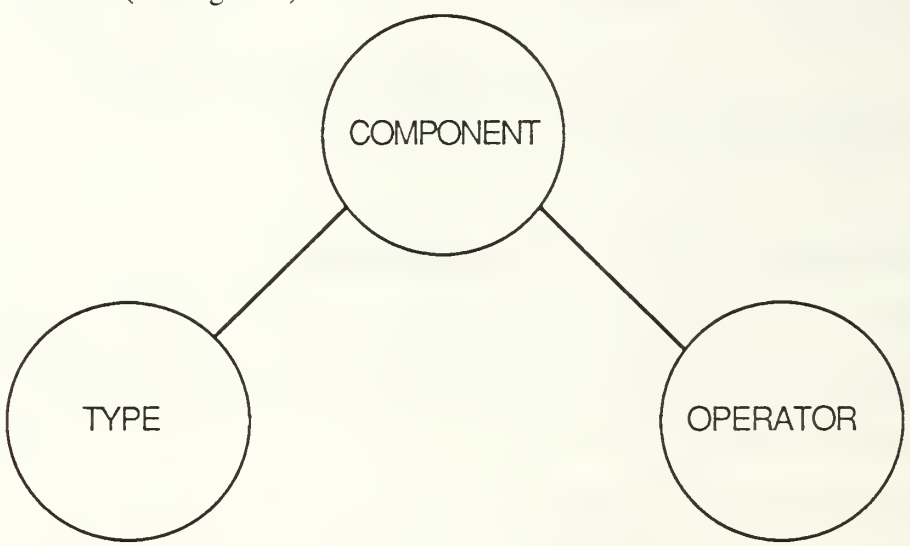

Figure 5. CAPS Software Base Class Hierarchy

a. Object Properties in the Software Base

The properties of the objects in the CAPS software base serve to uniquely identify each of the objects. The properties also can be used to model the state of an object as they describe the associations or connections between the objects. The properties of the objects are the PSDL specifications that define the class of operators or types. One of the properties for each operator or type is the Ada code module associated with the PSDL specifications. One of the major characteristics of object-oriented systems is the concept of inheritance. In general, if all the subclasses of a class define the same property, then that

property should be "factored out" to the superclass. The superclass will define the property so the subclasses can inherit it. Inheritance provides for the easy creation of objects that are almost like other objects. Inheritance reduces the need to specify redundant information and simplifies updating and modification, since the information only has to be entered and changed in one place. In the CAPS software base, the properties "name" and "generic" are two properties that can possibly be assigned to the Component class. Each of these values can then be inherited by the subclasses, Operators and Types. A type definition language called TDL, which is part of the Vbase Object-Oriented Database Management System [Ref. 30], will be used at the end of this section of the thesis to illustrate the classes and properties discussed thus far.

b. Object Methods in the Software Base

The next step in schema formation is to identify the frequent operations performed on the objects. To perform operations on the objects in an object-oriented system, messages must be sent by the application programmer to initiate an object's methods. Objects can also send messages to other objects to initiate action. Objects respond to messages using their methods to perform operations on the object. While properties are used to model the relationships between objects, the methods are used to access or modify the relationships and properties. A designer of an object-oriented system has the capability to model any complex form of object manipulation that may be desired. This is an extremely powerful capability and is one of the most favorable aspects of an object-oriented system. This thesis will identify some of the simple and complex methods that will be necessary to operate a productive Software Base Management System for CAPS.

The types of object operations that will be required for CAPS are those that will assist the prototype designer in quickly locating a suitable component for use in the

rapid prototyping process. Due to the fact that exact matches of PSDL specifications with a software base component will be rare, the designer needs assistance from the Software Base Management System to find alternate components that meet some of the specified designer requirements. The ideal situation is for the Software Base Management System to return a prioritized listing of components that the designer can then browse to make a selection of the one that most nearly satisfies the requirements. This type of operation is possible to program in an object-oriented system and will be demonstrated later in the implementation section of this thesis. Some of the simple and complex operations that are being proposed for the CAPS Software Base Management System are listed below:

- Operations to display an object's properties
- Operations to perform a search through a class of objects, looking for specific property values.
- Operations to perform an analysis of object property values, such as evaluating the number and types of input and output parameters associated with a PSDL specification
- Operations to create and delete instances of an object
- Operations to add, delete, or modify the property values associated with an instance of an object
- Operations that will establish a relationship between objects such as a "part of" relationship
- Operations that will search through instances of a class and return components that only partially match the specified designer requirements

Future enhancements and the addition of more complex operations can easily be performed by the Software Base Management System designer. It would involve only the re-compilation of the software that implements the schema in an object-oriented language or system.

c. Using An Object-Oriented Language to Define the Software Base Schema

To illustrate how a CAPS software base object can be defined in an object-oriented system, an example using the Vbase Type Definition Language (TDL) is provided below:

```
define type Operator

    supertypes = {Component};

properties =
    {
    Name : String;
    Inputs : distributed set[Input];
    Outputs : distributed set[Output];
    Generic : Boolean;
    Max_Execution_Time : Real;
    Max_Response_Time : Real;
    Min_Calling_Period : Real;
    Initial_State : Real;
    Period : Real;
    Description : String;

    Ada_Code : String;
    };

operations =
    {
    Build_Display (op : Operator)
        returns (Operator)
        method (OperatorBuildDisplay);
    Find_Exact_Match (op : Operator, name: String)
        returns (Operator)
        method (OperatorFindExactMatch);

    Find_Best_Match (op : Operator, name : String,
                     MET : Real)
        returns (Operator)
        method (OperatorFindBestMatch);
    };
end Operator;
```

The declaration of the object class for "Operator" listed above demonstrates how the properties and some of the methods for an object would be declared. Note that the properties "Inputs" and "Outputs" are declared as sets containing objects. These input and output objects are further defined by the declarations below. Not shown here, but recommended for future implementation is a method which will take the sets of inputs and outputs and perform an evaluation to determine if there are partial matches with the designer's specified requirements. Also note that the properties inputs and outputs could also have been declared to be of type string.

```
define type Input

properties =
      {
      InputName : String;
      Input_Type : Input_Output_Type;
      };

end Input;

define type Output

properties =
      {
      OutputName : String;
      Output_Type : Input_Output_Type;
      };

end Output;

define type Input_Output_Type

      is enum (String_Type, Integer_Type, Real_Type,
                  Boolean_Type);
```

Note in the declaration of the class "Types" below, the property attribute "IsOperator" is defined in terms of another object, the class of objects "Operator," which was declared above. This demonstrates some of the flexibility of object-oriented systems.

```
define type Types

supertypes = {Component};

properties =
      {
      Name : String;
      Generic : Boolean;
      Description : String;
      IsOperator : Operator;
      };
end Types;
```

The implementation of some of the above methods is described in the next section of this thesis. To activate any of the methods, the user must send a message to the object. The details of exactly how the operation is performed is hidden from the user. This demonstrates the concept of abstraction. The user doesn't need to know the details of how the Software Base Management System returns the best possible component match from the software base. The details of how it is done is a decision that is properly left to the designer of the object-oriented system. As priorities change, this designer has the option of writing new, improved methods to implement different operations on the objects as necessary.

Although the software base may contain many modules within a class with the same name, the PSDL specifications will serve to uniquely identify each module. Queries to the database can be structured to select the object modules having the desired PSDL specification values that most nearly meet the designer's requirements. This schema based on an object-oriented data model provides an expressive tool for representing the reusable Ada software components and also has the advantages of understandability and simplicity.

D. IMPLEMENTATION OF AN OBJECT-ORIENTED SOFTWARE BASE MANAGEMENT SYSTEM FOR CAPS USING VBASE

In this section of the thesis, some of the implementation details for the CAPS Software Base Management System will be demonstrated. But first, an overview will be given of the Vbase object-oriented database management system, which is the system that is recommended for use within CAPS. Vbase is a product of Ontologic Inc. It can be run in the Unix environment on Sun workstations, which makes it particularly attractive for CAPS. The "Vbase Integrated Object System" [Ref. 30], is a database and language platform for rapidly and inexpensively building sophisticated complex data and process intensive applications. Ontologic states that integrated object systems reflect the convergence of work in artificial intelligence, object-oriented programming languages, and databases. They also state that these systems combine the key aspects of each to provide a dramatic improvement in software programming platforms as object systems will emerge as a new standard in the near future. Not only can a Vbase application reference internal data and code; it can also reference existing external data and code. In this way, Vbase can be used to integrate several existing applications and databases into a single application with a common model. This is a particularly attractive capability that can be put to use in the integrated CAPS environment and it will be demonstrated later in this section.

1. Vbase System Overview

The Vbase system environment consists of the following components: (see Figure 6)

- Vbase Database: Persistent objects are stored in the Vbase database. Objects can be shared among multiple processes concurrently; access control and backup/recovery facilities are provided.
- Type Definition Language (TDL): This is the Vbase data definition language. TDL is used to define object types in the database, along with object properties, operations, and inheritance. Its role is similar to that of the data definition languages

of traditional databases. It creates a typing structure in the database to serve as a data model or conceptual schema for applications.

- "C" Object Processor (COP): This is the Vbase data manipulation language. COP is an object extension of Kernighan and Ritchie standard C. It is used to implement the operations of the object types defined using TDL.
- Integrated Toolset (ITS): a debugging and application tool environment that runs on top of Unix.
- Object SQL: Vbase's query facility providing the retrieval features of the SQL relational query language with extensions relevant to an object database system.

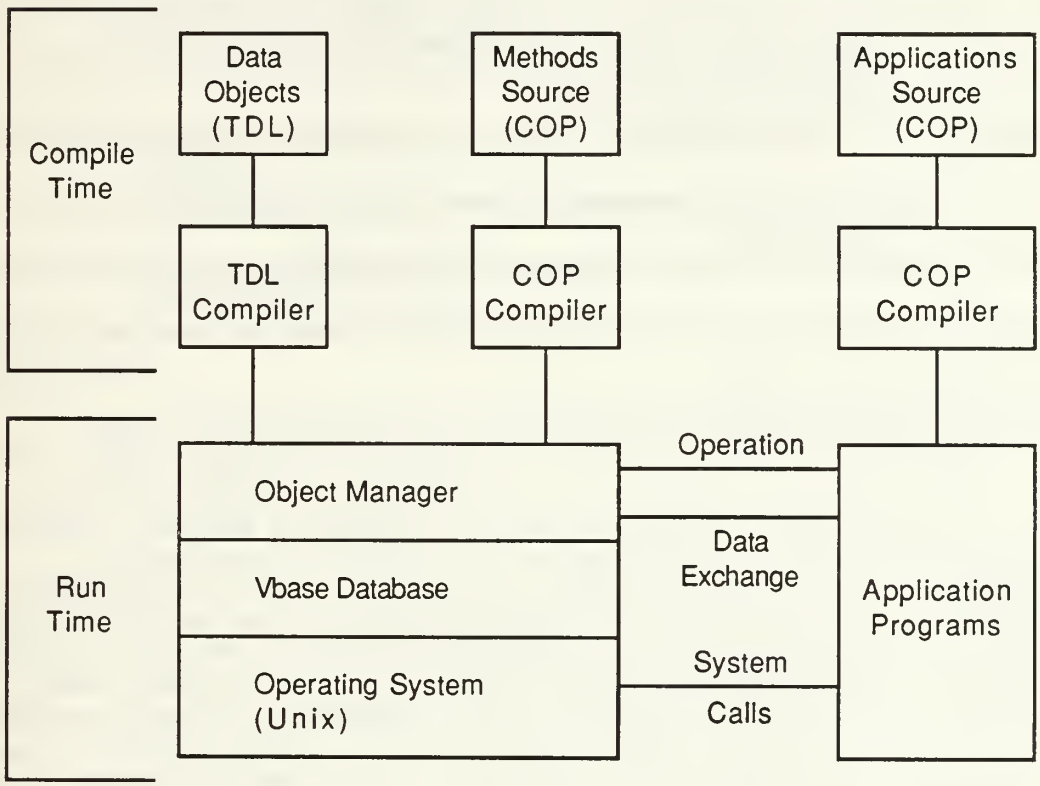

Figure 6. Vbase Components and Connections

2. Vbase Database Design

The steps involved in a typical Vbase database design are listed below:

- Identify the objects
- Identify their properties as much as possible
- Identify the frequent operations performed on the objects
- Define the objects using TDL
- Compile and debug the TDL definition of the objects

- Develop COP routines which implement the operations
- Compile and debug the COP programs

This methodology was utilized in the last section in the identification of the objects in the CAPS software base. TDL was used to define the objects, classes, properties and operations on the objects. The TDL code illustrates the encapsulation ability of Vbase, which separates the external interface of an object from its internal workings. To achieve encapsulation, each object type is made up of a separate specification and implementation. TDL is used to specify the specification, which defines the interface of the object (the operations which can be performed upon it) and the structure of the object (its attributes or properties and its relationships with other objects). The implementation is the actual COP computer code and data structures that cause the object to behave according to its specification. Users can only interact with the object through the operations defined in the specification written in TDL.

The TDL code below illustrates an example of creating an instance in the database. Note that this is only one of the ways to load the database. Application programs can be written to read input data, which would provide a faster mechanism for loading the database. These programs would make use of methods written by the software base designer to allow for the creation, deletion, and modification of objects in the software base. Vbase already has built-in features within COP which aid in object creation, deletion, and modification. The TDL code below creates an instance of an Operator in the software base. For simplicity, the inputs and outputs properties are declared to be of type string. When the Ada code from this object is printed to the terminal or a file, it is printed in the proper multi-line format.

```
define Operator Operator_1

    name = "start_up";
    inputs = "patient_chart: medical_history, temperature:  real";
    outputs = "estimated_power: real, treatment_finished:  boolean";
```

```
    max_execution_time = 90;

    description = "{ Extracts the tumor diameter from the
                  medical history and uses it to calculate
                  the maximum safe treatment power.
                  Estimate power is zero if no tumor is present.
                  Treatment finished is true only
                  if no tumor is present. } ";


ADA_CODE = "   WITH weapons_controller_package;
               USE weapons_controller_package;
               PROCEDURE controller_start_up (control_panel : IN real;
                                              sensor_input : IN real;
                                          missile_control : OUT real) IS
                enable_signal : real;
                BEGIN
                  enable_signal := calculate_control_level (control_panel,
                                                            sensor_input);
                  missile_control := enable_signal;
               END controller_start_up; ";

end Operator_1;
```

3. Using COP to Implement the Methods Associated With the Objects in the CAPS Software Base

The final phase in the design of an object-oriented system is to implement the operations or methods which have been defined for the objects and classes in the software base. These methods were declared within the objects using TDL code in the previous section. Each of the methods must now be implemented using COP. COP is an object-based superset of the language C. All features and control structures of standard C will work within Vbase. COP provides extensions to C to allow for operations involving the objects in the database. This is a very powerful capability of Vbase as the full capabilities of the C programming language are available to the software base designer and applications programmer. One of the methods that was defined for the object "Operator" was "OperatorFindExactMatch." This operation takes as input the name of an operator and iterates through the instances in the class of operators to find an exact match. This is a

very simple example that was chosen just to demonstrate the concept and feasibility. The method could just as easily have been written to check for a match of all the PSDL specifications input by the user. The COP code below implements the method "OperatorFindExactMatch" and illustrates a search based on operator name only:

```
#include <stdio.h>      /* include standard C routines */

#include <string.h>

#define MAXSIZE 500
import $Type;           /*  $ denotes a Vbase object   */
import $Class;          /*  import these objects needed for   */
import $Operator;       /*  this application                  */

enter module $Operator;   /*  use the object called operator  */

method
obj $Operator
OperatorFindExactMatch (anOperator,inputString)

obj $Operator anOperator;
obj $String inputString;

{               /* declaration of local variables */
      FILE *out;
      char op_name[MAXSIZE];
      char op_input[MAXSIZE];
      char op_output[MAXSIZE];
      char op_description[MAXSIZE];
      char op_ADA_CODE[MAXSIZE];
      int found = 0;
      obj $Operator theOperator;
      obj $Entity theEnt;
      obj $Integer theMET;

      out = fopen("SB_OUT,"  "w");

      iterate (theEnt = $Operator.class)     /*  iterate through the Operator class   */
      {
        theOperator = assert (theEnt, obj $Operator);
        if (theOperator.name == inputString)
        {
          found = 1;
            /* convert object code to C code */
          AM_stringToC(theOperator.name, op_name, MAXSIZE);
```

```
        AM_stringToC(theOperator.input, op_input, MAXSIZE);
        AM_stringToC(theOperator.output, op_output,
                              MAXSIZE);
        AM_stringToC(theOperator.description,
                op_description, MAXSIZE);
        AM_stringToC(theOperator.ADA_CODE, op_ADA_CODE,
                                  MAXSIZE);

        printf("PSDL SPEC NAME:   %s\n," op_name);
        printf("PSDL SPEC INPUT:  %s\n," op_input);
        printf("PSDL SPEC OUTPUT: %s\n," op_output);
        printf("PSDL SPEC MET:    %d\n,"
            theOperator.max_execution_time);
        printf("PSDL DESCRIPTION: %s\n");
        printf("%s\n," op_description);
        printf("                  %s\n");
        printf("ADA CODE:         %s\n");
        printf("%s\n," op_ADA_CODE);

        fputs (op_ADA_CODE, out);
        printf("         %s\n");
      }
    }
    if (found != 1)
      {
       printf("NO MATCH IN THE SOFTWARE BASE %s\n");
      }

  return (anOperator);

}
```

The COP code below implements the method "OperatorFindBestMatch." It is very similar to the code for OperatorFindExactMatch, except that it allows for the case where there are two operators in the software base with the same name. OperatorFindBestMatch returns the operator (or operators, if there is more than one) that have maximum execution times less than the user specified value. The criteria as to what exactly constitutes a "best match" is a decision that must be made and implemented by the designer of the Software Base Management System. In this simple example, the maximum execution time for an operator was chosen as the primary factor in determining component suitability.

```
method
obj $Operator
OperatorFindBestMatch (anOperator, inputString, theMET)

obj $Operator anOperator;
obj $String inputString;
obj $Integer theMET;

{           /* declaration of local variables */
        FILE *out;
        char op_name[MAXSIZE];
        char op_input[MAXSIZE];
        char op_output[MAXSIZE];
        char op_description[MAXSIZE];
        char op_ADA_CODE[MAXSIZE];
        int found = 0;
        obj $Operator theOperator;
        obj $Entity theEnt;

        out = fopen("SB_OUT,"  "w");

        iterate (theEnt = $Operator.class)
        {
          theOperator = assert (theEnt, obj $Operator);
          if ((theOperator.name == inputString) &&
            (theOperator.max_execution_time < theMET))
          {
            found = 1;
              /* convert object code to C code */
            AM_stringToC(theOperator.name, op_name, MAXSIZE);
            AM_stringToC(theOperator.input, op_input,
                                MAXSIZE);
            AM_stringToC(theOperator.output, op_output,
                                  MAXSIZE);
            AM_stringToC(theOperator.description,
                    op_description, MAXSIZE);

            AM_stringToC(theOperator.ADA_CODE, op_ADA_CODE,
                                    MAXSIZE);

            printf("PSDL SPEC NAME:    %s\n,"  op_name);
            printf("PSDL SPEC INPUT:   %s\n,"  op_input);
            printf("PSDL SPEC OUTPUT:  %s\n,"  op_output);
            printf("PSDL SPEC MET:     %d\n,"
                  theOperator.max_execution_time);
            printf("PSDL DESCRIPTION:  %s\n");
            printf("%s\n,"  op_description);
            printf("                   %s\n");
            printf("ADA CODE:          %s\n");
            printf("%s\n,"  op_ADA_CODE);
```

```
            fputs (op_ADA_CODE, out);
            printf("        %s\n");
            }
        }
        if (found != 1)
            {
             printf("NO MATCH IN THE SOFTWARE BASE %s\n");
            }

        return (anOperator);
}
```

The other operations and methods which were discussed in the previous section have not been implemented in this thesis. An excellent follow-up thesis to this one can concentrate on implementing all the software base object methods. The intention of this thesis was just to demonstrate the feasibility of using an object-based system for the CAPS Software Base Management System.

4. Interface Requirements Between the SBMS and the User Interface

From within the CAPS User Interface module, the designer will select the option of gaining access to the SBMS. To activate the SBMS, the User Interface module will call a COP application program called "Software_Base." This program (see Appendix B) serves as the primary query application routine that will send messages activating the methods that will perform the queries and other software base operations. It serves as the interface between the prototype designer and the underlying system (Vbase). The goal is to make the underlying system (Vbase) transparent to the designer. In other words, the designer should not need to know any details about Vbase in order to make use of the CAPS SBMS.

The COP application program will read the file containing the PSDL specifications. This file is called "PSDL_SPECS," and is provided from the User Interface by the Syntax Directed Editor. It serves as the designer's input to the SBMS. Using the specifications input by the designer, the appropriate methods will be activated to search the software base for exact and best matches that will meet the designer's

requirements. The output from the SBMS is the Ada code module displayed on the screen, which is also written to an output file called "SB_OUT." The designer now has the option of making use of that module or returning to the User Interface to further decompose his PSDL program. To get back into the SBMS, the designer will again have to select that option from within the User Interface.

The search of the software base can be made based on the object name, and is more completely defined through the PSDL specifications. The probability of an exact match is most likely not very high. Therefore, some flexibility must be provided within the COP application routines in order to locate near-matches in an attempt to find a possibly useful alternative Ada module.

5. Other Useful Vbase Capabilities

Future enhancements to the CAPS SBMS should provide an interface to enable the prototype designer to make use of the additional Vbase capabilities discussed below. One of these added capabilities is the Integrated Tool Set (ITS). The ITS is a single tool combining the functionality of a source browser, database browser, and source debugger. The source browser is used to view COP source files. The debugger is a source-level COP debugger used to control a running Vbase application program. The database browser is used to display and modify application database objects. The database browser thus provides the designer with some flexibility as an alternate means to view the objects in the software base.

Another option available to the prototype designer is the use of Vbase Object SQL. Object SQL is the Vbase implementation of the SQL query language. It is an interactive database query language used to generate tabular reports. Reports can be displayed on the screen or copied to files for further processing. Object SQL is a fourth generation language which should be easy to learn for those having any familiarity with

relational SQL. It is simple and convenient enough to be used for casual queries of the database. It is also powerful enough to produce many of the data processing reports that would otherwise require conventional programming. Short simple queries are generally typed in on the fly giving the user interactive access to the database.

6. Problems With Vbase

Vbase and the other OODBMS systems currently available are still very new and have a number of shortfalls that need to be addressed. One of the current major problems with Vbase is the quality of the user documentation. The user's manual has a number of errors in it including sample programs that will not run due to syntax errors. These were pointed out to Ontologic Inc. and updates to the manual are in progress. A listing of some of the syntax errors and user manual problems is available in Appendix C.

Vbase has a very slow compilation speed. The process of defining the objects and the database schema through the use of TDL and COP is a slow, time-consuming process. This problem is even more pronounced with more complex schemas. Once the TDL and COP programs are successfully compiled, the actual runtime of the programs in simple applications appears to be adequate.

The objects in the CAPS software base contain Ada program modules as one of the properties of the object. In the implementation of the software base, great difficulties were encountered in loading the software base with multi-line text input. A Vbase database can be loaded either through the use of TDL code as illustrated in this thesis or through the use of a COP method. It was found that the same multi-line text input would work using TDL but not when using COP. The COP compiler had problems identifying a carriage return. This problem was reported to Ontologic, and they are investigating it.

Vbase does not have a capability to easily perform schema modification, which is supposed to be one of the advantages of object-oriented database management systems.

Modifications to the Vbase database schema must be made through the use of TDL and COP code, which then has to be recompiled. This is a time-consuming process. Vbase also has some problems with inheritance operations and does not support multiple inheritance. The vendor claims that these are all future enhancements that remain to be implemented. Object SQL appears to have been added to Vbase due to market demand and familiarity with SQL, rather then to any sound technical basis.

Typing a carriage return during a multi-line text input will result in a Vbase error message. In order to make a multi-line text input into a Vbase database, the user can either use a continuous string bounded by quotation marks or use the sequence of characters "\412" in place of a carriage return.

Many of the problems encountered in the development of the CAPS software base are typical of the problems that are experienced with any new software system. As the Vbase system matures, eventually these type of problems should be overcome.

E. DISCUSSION OF KNOWLEDGE BASE SUPPORT FOR RAPID PROTOTYPING

This section will discuss the possibility for the inclusion of expert system technology into the CAPS SBMS. As presented in Ref. 31, it may be desirable to automate the process of managing the reusable components in the software base because this frees the human designer from the burden of remembering what software components are currently available. Expert system technology is appropriate for the problem because the number of useful software components is so large that it is impractical to store them all explicitly. The retrieval of reusable components from a software base is a difficult search problem in a very large space. Exhaustive searching is impractical because the space is much too large. These considerations suggest using heuristic search methods to aid in the retrieval of software modules based on approximate matches of PSDL specifications. Heuristic search

methods are a common component of expert systems. Expert system technology can be more effective and produce more accurate automated retrievals than classical information retrieval techniques using keyword searching. Further research into the incorporation of expert system technology within the CAPS SBMS is warranted as it will further contribute to software designer productivity. This would be an excellent topic for a follow-up thesis.

IV. CONCLUSIONS AND RECOMMENDATIONS

A. SUMMARY AND CONCLUSIONS

There are very few tools currently available that help the designers and users of hard, real-time software systems to clearly define and analyze critical system requirements and timing constraints. The Computer Aided Prototyping System is a software engineering tool that is currently being developed that implements the rapid prototyping methodology in the design of hard, real-time, embedded software systems. CAPS provides a flexible and powerful tool that offers a new approach to the development of software in a cost-efficient and timely manner. The most noteworthy contribution of CAPS to the advancement of the rapid prototyping methodology lies in the fact that it provides for the automatic generation of executable prototypes through the use of formal specifications and reusable software components.

This thesis has concentrated on the development of the Software Base Management System, which is a key component of the CAPS system and the rapid prototyping methodology. A robust Software Base Management System containing a library of reusable Ada components can significantly contribute to the success of the rapid prototyping process and enhance designer productivity.

The goal of this thesis was to develop a conceptual level design of the Software Base Management System for CAPS, and to provide guidelines for its implementation. A basic structure for the Software Base Management System was developed using object-oriented technology, and a simple implementation was demonstrated using a new database software package (Vbase). The advantage of using an object-oriented system is that it offers the designer a high level of flexibility and power in implementing the most complex database

operations. An object-oriented approach offers increased modeling power, a high level of abstraction, and the ability to inherit and refine object properties. Interface requirements and integration within the CAPS prototyping environment were also discussed. This study has accomplished the goals of the thesis and identified key aspects of the Software Base Management System for future implementation and possible follow-up thesis work.

B. RECOMMENDATIONS FOR FURTHER RESEARCH

This thesis has provided a foundation upon which further implementation of the Software Base Management System can be based. Further research and testing is required to complete full implementation of the system and to identify potential weaknesses. This author recommends work in the following specific areas:

- The design and implementation of methods within the software objects to perform more complex analysis and software base retrieval operations. Some of the methods that were proposed for future implementation include the analysis of the type and number of input and output parameters. It is recommended that a metric be developed to assist in this analysis and provide a standard upon which to evaluate the suitability of components having only a partial match with the specifications.
- Implementation of methods to assist the prototype designer in performing software base maintenance operations. These operations include the insertion of new modules into the software base, as well as provide for the deletion or modification to existing object modules.
- Design and implementation of a CAPS term rewrite system which will provide for the normalization of software specifications to be used to find and retrieve the required components from the software base. This is perhaps the most difficult subsystem of CAPS that needs to be implemented. Because there are many syntactic forms for the same semantic description, reduction to a normal form is a necessity as it will provide a more practical approach than trying to generate all the different possible variations of a specification description and searching the software base for each variation.
- The development of an expert system interface to the software base. This system can possibly include some of the term rewrite operations and can also assist in the analysis of component suitability in matching the prototype designer's specifications.

APPENDIX A

MAPPING OF PSDL TO ADA

PSDL SPECIFICATION:

```
OPERATOR start_up
SPECIFICATION
 INPUT patient_chart: medical_history, temperature: real
 OUTPUT estimated_power: real,  treatment_finished : boolean
  BY REQUIREMENTS start_up time
 MAXIMUM EXECUTION TIME 90 ms
  BY  REQUIREMENTS temperature_tolerance
 DESCRIPTION
 { Extracts the tumor diameter from the medical history and uses
it to calculate the maximum safe treatment power.  Estimated
power is zero if no tumor is present.  Treatment finished is true
only if no tumor is present.
 }
END
```

ADA IMPLEMENTATION:

```
WITH medical_history_package; USE medical_history_package;
PROCEDURE start_up (patient_chart : IN medical_history;
                    temperature: IN real;
                    estimated_power: OUT real;
                    treatment_finished: OUT boolean) IS
 diameter: real;
 k: constant real := 0.5;
BEGIN
 diameter := get_tumor_diameter(patient_chart, "brain_tumor");
 estimated_power := k * diameter * 2;
 treatment_finished := false;
EXCEPTION
 WHEN no_tumor =>
  estimated_power := 0.0;
  treatment_finished := true;
END start_up;
```

PSDL SPECIFICATION:

```
OPERATOR maintain
 SPECIFICATION
 INPUT temperature: real
```

```
  OUTPUT estimated_power: real, treatment_finished: boolean
  MAXIMUM EXECUTION TIME 90 ms
    BY REQUIREMENTS temperature_tolerance
  DESCRIPTION
  { The power is controlled to keep it between 42.4 and 42.6
degrees C.
  }
END
```

ADA IMPLEMENTATION:

```
PROCEDURE maintain (temperature: IN real;
                    estimated_power: OUT real;
                    treatment_finished: OUT boolean) IS
  c: constant real := 10.0
BEGIN
  IF temperature > 42.5 THEN estimated_power := 0.0;
  ELSE estimated_power := c * (42.5 -temperature);
  END IF;
  treatment_finished := true;
END maintain;
```

PSDL SPECIFICATION:

```
OPERATOR safety_control
SPECIFICATION
  INPUT treatment_switch, treatment_finished : boolean
        estimated_power: real
  OUTPUT treatment_power: real
    BY REQUIREMENTS shutdown
  MAXIMUM EXECUTION TIME 10 ms
    BY REQUIREMENTS temperature_tolerance
  DESCRIPTION
  { The treatment power is equal to the estimated power if the
treatment switch is true and treatment finished is false, and
otherwise the treatment power is zero,
  }
END
```

ADA IMPLEMENTATION:

```
PROCEDURE safety_control (estimated_power: IN real;
                          treatment_finished: IN boolean;
                          treatment_switch: IN boolean;
                          treatment_power: OUT real) IS
BEGIN
  IF treatment_switch and not treatment_finished
  THEN treatment_power := estimated_power;
  ELSE treatment_power := 0.0;
```

```
  END IF;
END safety_control;
```

APPENDIX B

SAMPLE APPLICATION PROGRAM

```
#include <stdio.h>
#include <string.h>

import $Operator;


/*  THIS APPLICATION PROGRAM READS IN SELECT PSDL          */
/*  SPECIFICATIONS FROM AN INPUT FILE AND CALLS THE        */
/*  VBASE METHODS TO PERFORM QUERIES ON THE SOFTWARE BASE.  */


void substring (instr, startpos, numchar, substr)
/*  THIS FUNCTION TAKES AN INPUT STRING AND RETURNS        */
/*  A SUBSTRING                                            */
char instr[30], substr[30];
int startpos;
int numchar;
{
    if (startpos < strlen(instr))
        {
            instr[0] = '\0';
           strcat (instr, instr + startpos);
        }
    strncpy (substr, instr, numchar);
}


main (argc, argv)

int argc;
char **argv;

{
   int eof_flag = 1;
   int integer_MET;
   char line[30];
   char operatr[30], MET[30];
   char operator_name[30], met[30];
   FILE *in;

   char *dbname;
   char *getenv();
```

```
obj $Operator theOperator, currentOperator;
obj $String theOperatorName;
obj $Integer theMET;

in = fopen("PSDL_SPECS,"  "r");
if (argc > 1)
  {
  dbname = argv[1];
  }
else if (dbname = getenv("DBNAME"))
  {
  }
else
  {
  printf("Must specify database name, either as a command
line argument,\or via the Unix environment variable DBNAME\n");
  exit(1);
  }

{
 fgets(line, 30, in);

 while (feof(in) != eof_flag)
   {
    strncpy (operatr, line, 8);
    if (strncmp("OPERATOR,"  operatr, 8) == 0)
    {
       substring(line, 9, (strlen(line) - 10),
                        operator_name);
     }

    strncpy (met, line, 22);
    if (strncmp("MAXIMUM EXECUTION TIME,"  met, 22) == 0)
    {

      substring (line, 23, (strlen(line) - 24), MET);
      integer_MET = (atoi(MET));

    }
      fgets (line, 30, in);
   }


AM_databaseOpen (dbname, 0);   /* OPENS DATABASE */

   theOperatorName = operator_name;

   theMET = integer_MET;

   (void) $Operator$FindExactMatch ($Operator_1,
                    theOperatorName);
```

```
    (void) $Operator$FindBestMatch ($Operator_1,
                theOperatorName, theMET);
  }
  protect
  AM_databaseClose (dbname);

}
```

APPENDIX C

LISTING OF ERRORS IN VBASE MANUAL

For manual issued September 29, 1987

1. In the "Tutorial Program Stage 2", the file called PartsExceptions.tdl has a syntax error in the first line of the file:

"define Exception PartException" should read "define ExceptionType PartException."

2. On page III-12, there is a syntax error in the sample program. The built-in function "AM_StringToC" should read "AM_stringToC."

3. On page III-18, there is a syntax error in the sample program. The built-in function "AM_StringToC" should read "AM_stringToC."

4. On page III-74, there is an error in the sample program. The semicolon should be removed after the iterate statement to allow the operations after the statement to be performed in the iterate loop.

LIST OF REFERENCES

1. Elbert, T. F., *Embedded Programming in Ada*, Van Nostrand Reinhold Publishing Co., Inc., New York, NY, 1986.

2. Chitwood, G., "Ada Meets the Challenge of Real-Time Simulation," *Defense Computing*, v. 1, no. 4, pp. 32-38, July/August 1988.

3. Nielsen, K., and Shumate, K., *Designing Large Real-Time Systems With Ada*, Intertext Publications/Multiscience Press, Inc., New York, NY, 1988.

4. Winograd, T., "Beyond Programming Languages," *Communications of the ACM*, pp. 391-401, July 1979.

5. Barstow, D. R., Shrobe, H. E., and Sandewall, E., *Interactive Programming Environments*, McGraw-Hill Book Company, New York, NY, 1984.

6. Luqi and Ketabchi, M., *A Computer Aided Prototyping System*, Tech. Rep. NPS52-87-011, Naval Postgraduate School, Monterey, CA, 1987 and in IEEE Software, pp. 66-72, March 1988.

7. Luqi, *Rapid Prototyping for Large Software System Design*, Ph.D dissertation, University of Minnesota, Minneapolis, MN, 1987.

8. Berzins, V., and Luqi, *Software Engineering with Abstractions: An Integrated Approach to Software Development Using Ada*, Addison-Wesley Publishing Co., Inc., 1988.

9. Berzins, V., and Luqi, *Handbook of Computer-Aided Software Engineering: Languages for Specification, Design, and Prototyping*, Van Nostrand Reinhold Publishing Co., Inc., New York, NY, 1988.

10. Booch, G., *Software Engineering with Ada*, 2nd ed., Benjamin Cummings Publishing Co., Inc., Menlo Park, CA, 1987.

11. Prieto-Diaz, R., and Freeman, P., "Classifying Software for Reusability," *IEEE Software*, pp. 6-16, January 1987.

12. Yin, W., Tanik, M., Yun, D., Lee, T., Dale, A., "Software Reusability: A Survey and a Reusability Experiment," *Proceedings of the 1987 Fall Joint Computer Conference*, Computer Society Press of the IEEE, Washington D.C., 1987.

13. Burton, B. A., "The Reusable Software Library," *IEEE Software*, pp. 25-32, July 1987.

14. Goguen, J. A., "Reusing and Interconnecting Software Components," *Computer*, pp. 16-28, February 1986.

15. Matsumoto, Y., "A Software Factory: An Overall Approach to Software Production," *Tutorial: Software Reusability*, by Peter Freeman, Computer Society Press of the IEEE, Washington D.C., 1987.

16. Freeman, P., "Reusable Software Engineering Concepts and Research Directions," *ITT Proceedings of the Workshop on Reusability in Programming*, pp. 2-16, 1983.

17. Rice, J. R., and Schwetman, H. D., "Interface Issues in a Software Parts Technology," *ITT Proceedings of the Workshop on Reusability in Programming*, pp. 129-137, 1983.

18. Johnson, R. E., and Foote, B., "Designing Reusable Classes," *Journal of Object-Oriented Programming*, pp. 22-35, June/July, 1988.

19. Meyer, B., *Object-Oriented Software Construction*, Prentice Hall Inc., Englewood Cliffs, NJ, 1988.

20. Wegner, P., "Varieties of Reusability," *ITT Proceedings of the Workshop on Reusability in Programming*, pp. 30-44, 1983.

21. Parnas, D. L., "Enhancing Reusability with Information Hiding," *ITT Proceedings of the Workshop on Reusability in Programming*, pp. 240-247, 1983.

22. Booch, G., *Software Components with Ada*, Benjamin Cummings Publishing Co., Inc., Menlo Park, CA, 1987.

23. Lenz, M., "Software Reuse through Building Blocks," *IEEE Software*, pp. 34-42, July, 1987.

24. Berard, E., *An Object Oriented Design Handbook for Ada Software*, EVB Software Engineering, Inc., Frederick, MD, 1985.

25. Felsinger, R. C., *Object Oriented Design* (Course Notes), Technology Training Corporation, Torrance, CA, June 1988.

26. Conn, R., *The Ada Software Repository and the Defense Data Network*, Zoetrope Publishing Co., Inc., New York, NY, 1987.

27. Anderson, C., "The CAMP Approach to Software Reuse," *Defense Computing*, pp. 25-30, September/October 1988.

28. Ketabchi, M., *Object Oriented Database Management Systems for Complex Data and Process Intensive Applications* (Course Notes), Santa Clara University, Santa Clara, CA, June 1988.

29. Robson, D., "Object-Oriented Software Systems," *Byte Magazine*, pages 74-86, August, 1981.

30. *Vbase Integrated Object Database User's Manual*, Ontologic Inc., Billerica, MA, 1987.

31. Luqi, "Knowledge Base Support for Rapid Prototyping" *IEEE Expert*, November 1988.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Office of Naval Research Office of the Chief of Naval Research Attn. CDR Michael Gehl, Code 1224 800 N. Quincy Street Arlington, Virginia 22217-5000	1
4.	Space and Naval Warfare Systems Command Attn. Dr. Knudsen, Code PD 50 Washington, D.C. 20363-5100	1
5.	Ada Joint Program Office OUSDRE(R&AT) Pentagon Washington, D.C. 20301	1
6.	Naval Sea Systems Command Attn. CAPT Joel Crandall National Center #2, Suite 7N06 Washington, D.C. 22202	1
7.	Office of the Secretary of Defense Attn. CDR Barber STARS Program Office Washington, D.C. 20301	1
8.	Office of the Secretary of Defense Attn. Mr. Joel Trimble STARS Program Office Washington, D.C. 20301	1
9.	Commanding Officer Naval Research Laboratory Code 5150 Attn. Dr. Elizabeth Wald Washington, D.C. 20375-5000	1

10. Navy Ocean System Center — 1
Attn. Linwood Sutton, Code 423
San Diego, California 92152-5000

11. National Science Foundation — 1
Attn. Dr. William Wulf
Washington, D.C. 20550

12. National Science Foundation — 1
Division of Computer and Computation Research
Attn. Dr. Peter Freeman
Washington, D.C. 20550

13. National Science Foundation — 1
Director, PYI Program
Attn. Dr. C. Tan
Washington, D.C. 20550

14. Office of Naval Research — 1
Computer Science Division, Code 1133
Attn. Dr. Van Tilborg
800 N. Quincy Street
Arlington, Virginia 22217-5000

15. Office of Naval Research — 1
Applied Mathematics and Computer Science, Code 1211
Attn. Mr J. Smith
800 N. Quincy Street
Arlington, Virginia 22217-5000

16. New Jersey Institute of Technology — 1
Computer Science Department
Attn. Dr. Peter Ng
Newark, New Jersey 07102

17. Southern Methodist University — 1
Computer Science Department
Attn. Dr. Murat Tanik
Dallas, Texas 75275

18. Editor-in-Chief, IEEE Software — 1
Attn. Dr. Ted Lewis
Oregon State University
Computer Science Department
Corvallis, Oregon 97331

19. University of Texas at Austin — 1
Computer Science Department
Attn. Dr. Al Mok
Austin, Texas 78712

19. Defense Advanced Research Projects Agency (DARPA) 1
Director, Naval Technology Office
1400 Wilson Boulevard
Arlington, Virginia 22209-2308

20. Defense Advanced Research Projects Agency (DARPA) 1
Director, Strategic Technology Office
1400 Wilson Boulevard
Arlington, Virginia 22209-2308

21. Defense Advanced Research Projects Agency (DARPA) 1
Director, Prototype Projects Office
1400 Wilson Boulevard
Arlington, Virginia 22209-2308

22. Defense Advanced Research Projects Agency (DARPA) 1
Director, Tactical Technology Office
1400 Wilson Boulevard
Arlington, Virginia 22209-2308

23. COL C. Cox, USAF 1
JCS (J-8)
Nuclear Force Analysis Division
Pentagon
Washington, D.C. 20318-8000

24. LTCOL Kirk Lewis, USA 1
JCS (J-8)
Nuclear Force Analysis Division
Pentagon
Washington, D.C. 20318-8000

25. U.S. Air Force Systems Command 1
Rome Air Development Center
RADC/COE
Attn. Mr. Samuel A. DiNitto, Jr.
Griffis Air Force Base, New York 13441-5700

26. U.S. Air Force Systems Command 1
Rome Air Development Center
RADC/COE
Attn. Mr. William E. Rzepka
Griffis Air Force Base, New York 13441-5700

27. Professor Luqi 1
Code 52LQ
Naval Postgraduate School
Computer Science Department
Monterey, California 93943-5000

Thesis
G13355 Galik
c.1 A conceptual design of a Software Base Management System for the Computer Aided Prototyping System.